Box 4
Folder 8

1 of 4

Hotline on object-oriented technology, 1989-1993

Adele Goldberg papers

Periodicals, technical papers, and articles

X5774.2010, *Box 4* 102739322

THE COMPUTER HISTORY MUSEUM
1 027 3932 2

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes***, I want to be plugged into the latest thinking, and developments in the object-oriented technology. Immediately enter me as a charter subscriber at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	2 years (24 issues)
☐ ~~299~~ $249	☐ ~~$598~~ $498

outside the U.S.—add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT—U.S.currency drawn on U.S. bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card #_____ Exp. date_____
Signature _____

Name_____
Company _____
Address _____
City _____ State_____ Zip_____
Phone_____

For immediate service—call the Hotline 212-972-7055 or Fax 212-370-4983

HOTLINE on
Object-Oriented Technology
The International Newsletter

310 Madison Avenue
Suite 503
New York, NY 10017

Premiere Issue

HOTLINE on Object-Oriented Technology

The International Newsletter

Premiere Issue

November, 1989 **Volume 1, Number 1**

Which Object-Oriented Language Should We Choose?

by Ken Auer

More and more people in the software development industry are beginning to believe that object-oriented technology is worth investigating. As products emerge that support the technology, each vendor will tell you why their product is better than the rest. It is difficult to evaluate the product offerings due to lack of experience with the technology.

To simplify making these decisions feature checklists are often used. However, simple checks don't provide enough information. Each desired feature must be weighed as to its importance, and each product must be weighed with respect to how well it fills these requirements. I'd like to address "check weight" in the context of evaluating object-oriented languages.

To illustrate each of the issues raised, I will use for comparison Smalltalk-80, Smalltalk/V, Objective-C, and C++. These languages are a representative sample of the variety of object-oriented languages available (pure vs. hybrid). Many of the points made about Objective-C and C++ are true of other hybrid languages (e.g. Object-Pascal). In the same way, many of the points made about Smalltalk apply to other "pure" languages (e.g. Actor). These criteria can be applied to any object-oriented language.

One of the biggest selling points of object-oriented programming is the productivity gains which can be realized through its use. The first three items we'll address, reusability, development environments, and memory management, directly affect development productivity.

Reusability

The fastest way to produce code which is well designed, written, and tested is to already have it. Object-oriented languages provide a framework which facilitates the ability to produce code that can be used over and over again. This reusable software is often called the class library. The larger these class libraries become, the less new code has to be written for each application. It has often been my experience that, with a mature class library, only 20% of the code for a new application will need to be written. At times this rate is even lower. I've noticed significant increases in my development productivity after adding as little as a single class to my repertoire.

In her recent keynote address to the attendees of SCOOP '89, Adele Goldberg (CEO of ParcPlace Systems) continually stressed the importance of reusability. She proposed that we need to start evaluating programmers by the amount of reusable code they produce and the amount of code they reuse, as opposed to the number of lines of code they produce.

continued on page 3

Contents

Feature:
Which Object-Oriented Language Should We Choose?
1

Thoughts from the Editor
2

Note from the Publisher
2

Analysis and Design:
Understanding Object-Oriented Analysis
6

Languages and Environments:
Object-Oriented Programming: Myths and Realities
9

Data Bases:
An Introduction to Object-Oriented Database Management Systems
11

The International Beat
13

Research, Education and Training
14

Calendar
14

Book Review:
Object-Oriented Analysis
15

Object-Oriented Systems: the Commercial Benefits
16

Company Showcase
17

Product News
22

Book Listing
23

Thoughts from the Editor

Welcome to the premiere issue of the Hotline on Object-Oriented Technology! We are witnessing a tremendous surge of interest in object-oriented technology and HOOT, as we affectionately call the newsletter, will keep you informed of the latest developments in the field.

The primary goal of HOOT is education: not education as in the technical details, but education as to the issues, developments, and commercial offerings that you need to know to keep abreast of the rapid changes that are occurring as OO technology enters the mainstream of computing.

Each month, our feature article will present information of general interest in some depth. This month's feature, *Which Object-Oriented Language Should We Choose?,* explores the issues involved in choosing an object-oriented language.

As regular features, we will present article in three major areas of the technology:

OO Analysis and Design;
OO Languages and Environments
OO Data Bases.

In this issue, *Understanding Object-Oriented Analysis,* presents one perspective on Object-Oriented Analysis; explaining how it differs from more traditional analysis techniques. *Object-Oriented Programming: Myths and Realities* is the first of a series of articles presenting the issues of object-oriented languages and environments. *An Introduction to Object-Oriented Database Management Systems* is the first of another short series designed to introduce the reader to object-oriented data bases and relate them to existing data base technologies.

The remainder of this first issue of HOOT covers a range of topics including OO training in England, news from Europe, a book review, product news, and a calendar of OO conferences. You will also find our Company Showcase which introduces you to many of the commercial firms that have entered the object-oriented marketplace.

The review of the book, *Object-Oriented Analysis,* is of particular note in that it uses the term "Object-Oriented Analysis" in a manner significantly different than the Shlaer and Mellor article. This multiple use of the term is indicative of the current state of object-oriented technology, with disagreements over basic terminology still part of the ongoing process of development. A major role of HOOT is to keep you informed of the resolution of this and similar issues.

A newsletter is not complete without you, the readers. We extend to you an invitation to comment on the articles and positions taken in HOOT. A 'Letters to the Editor' section will be added in the next issue. We encourage you to tell us what you would like to see so that we may continue to present what you want to read.

Again, welcome to the *Hotline on Object-Oriented Technology.*

Jeff McKenna
Editor

Letter from The Publisher

Welcome to the premiere issue of HOOT. This issue represents months of planning, traveling, researching, recruiting, cajoling and arm-twisting to develop the editorial direction of HOOT and subsequently assembling a first-rate team of well-informed writers.

Why was HOOT created? Object-oriented programming techniques have exploded in popularity over the past 2 years. Industry analysts estimate that U.S. revenue from O-O software will reach $170 million in 1990 and then double by 1992. This trend towards mainstream object-oriented usage creates a need for a quick and reliable forum that culls the latest news and ideas while linking together users, worldwide. HOOT's editorial mission is to fulfill this need—to keep you well informed of all aspects of this new technology.

HOOT is written to be read. We will provide regular columns on object-oriented technical issues such as analysis and design, languages and environments as well as data bases. You will get well thought out perspectives and no-where-else found information on insightful applications, education and training issues, the scoop on the international scene, project management techniques, and the low down on trends, issues and people. Be informed of the latest product releases—even as they are being announced. And read our savvy candid product reviews which will direct you to the most useful products.

Functional, dependable and timely. I invite you to subscribe now at the special Charter Subscription rate of $249 now offered and get plugged into the hotline. HOOT is your pathway into the minds of those who are pushing out the ever-expanding boundaries of this technology.

Richard P. Friedman
Group Publisher

HOTLINE on
Object-Oriented Technology
The International Newsletter

Jeff McKenna
EDITOR

ADVISORY BOARD

Jim Anderson
Digitalk, Inc.

Tom Atwood
Object Design, Inc.

Cort DeVoe
IBM

Adele Goldberg
ParcPlace Systems

Bertrand Meyer
Interactive Software Engineering

Reed Phillips
Knowledge Systems Corp.

Trygve Reenskaug
Taskon A/S

Dave Thomas
Object Technology International

Zack Urlocker
The Whitewater Group

Eugene Wang
Borland International

Richard P. Friedman
Group Publisher

Xavier Yaffar
Technical Advisor

Christine Coleman
Art Director

Melanie Younossi
Marketing Assistant

The Hotline on Object-Oriented Technology (ISSN#1044-4319) is published monthly by SIGS Publications Inc., 310 Madison Ave., Suite 503, New York, NY 10017. (212)972-7055. ©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher.

Mailed First Class. Subscription rate—1 year (12 issues) $299. Foreign and Canada $323. Single copy $30. Send address changes and subscription orders to HOOT, 310 Madison Avenue, Suite 503, NY, NY 10017
Submit editorial correspondance to Editor, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062

Book Listing

Here is a listing of OO books with author, publisher, ISBN, price and date published.

OBJECT-ORIENTED ENVIRONMENT IN C++
David Hu
MIS Press
ISBN 1-55826-014-6
$29.95—10/89

USER INTERFACES IN C++ AND OBJECT-ORIENTED PROGRAMMING
Mark Goodwin
MIS Press
ISBN 1-55828-023-5
$26.95—5/89

USING QUICK PASCAL
Steven Nomeroff
Osborn-McGraw Hill
IBSN 0-07-881520-7
$24.95—9/89

USING C++
Bruce Eckel
Osborn-McGraw-Hill
ISBN 0-07-881522
$24.95—7/89

TURBO-PASCAL 5.5—COMPLETE REFERENCE
Stephen K. O'Brien
Osborn-McGraw-Hill
ISBN 0-07-881501-0
$26.95—6/89

TUTORIAL: OBJECT-ORIENTED COMPUTING
VOLUME 1: CONCEPTS
VOLUME 2: IMPLEMENTATIONS
Gerald Peterson
EEE Comp. Soc. Press
Vol 1. ISBN 0-8186-0821-8 $40—1/88
Vol. 2 ISBN 0-8186-0822-6 $50—1/88

C++ FOR C PROGRAMMERS
Ira Pohl
Benjamin/Cummings
ISBN 0-8053-0910-1
$29.95—4/89

A C++ PRIMER
Stanley B. Lippman
Addison-Wesley
$30.25—5/89

THE C++ ANSWER BOOK
Tony Hansen
Addison-Wesley
$26.95

SMALLTALK-80 THE LANGUAGE
Adele Goldberg
& David Robson
Addison-Wesley
$31.25

OBJECT-ORIENTED CONCEPTS, DATABASES & APPLICATION
Won Kim &Frederick H. Lochovsky
Addison-Wesley
$43.25

OBJECT ORIENTED DESIGN WITH EXAMPLES IN C++
Mark Mullin
Addison-Wesley
ISBN 0-201-51722-1
$19.95—8/89

HANDS-ON C++ PROGRAMMING
Joseph McIsaac & William M. Miller
Addison-Wesley
ISBN 201-51816-3
$22.95—11/89

OBJECT-ORIENTED PROGRAMMING FOR ARTIFICIAL INTELLIGENCE: A GUIDE TO TOOLS AND SYSTEMS DESIGN
Ernest R. Tello
Addison-Wesley
ISBN 0-201-09228-X
$22.95—8/89

OBJECT-ORIENTED PROGRAMMING IN TURBO PASCAL 5.5
Ben Ezzel
Addison-Wesley
ISBN 0-201-52375-2
$24.95—1/90

AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING AND SMALLTALK
Lewis Pinson & Richard Wiener
Addison-Wesley
ISBN 0- 201-19127-X
$27.95

AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING AND C++
Richard Wiener & Lewis Pinson
Addison-Wesley
ISBN 0-201-15413-7
$27.95

A LITTLE SMALLTALK
Timothy A. Budd
Addison-Wesley
ISBN 0-201-10698-1
$29.25

OBJECT-ORIENTED PROGRAMMING: AN EVOLUTIONARY APPROACH
Brad Cox
Addison-Wesley
ISBN 0-201-10393-1
$32.25

THE C++ PROGRAMMING LANGUAGE
Bjarne Bjarne Stroustrup
Addison-Wesley
ISBN 0-201-12078-X
$30.25

SMALLTALK 80: THE INTERACTIVE PROGRAMMING ENVIRONMENT
Adele Goldberg
Addison-Wesley
ISBN 0-201-11372-4
$38.75

SMALLTALK 80: BITS OF HISTORY, WORD OF ADVICE
Glenn Krasner
Addison-Wesley
ISBN 0-201-11669-3
$30.25

THE COMPLETE C++ PRIMER
Keith Weiskamp & Bryan Flamig
Academic Press
ISBN 0-12-742687-6
$34.95—12/89

OBJECT ORIENTED SYSTEM ANALYSIS
Sally Shlaer & Stephen Mellor
Prentice Hall
ISBN 013-629023-x
$31.00—4/88

OBJECT ORIENTED SOFTWARE CONSTRUCTION
Bertrand Meyer
Prentice Hall
ISBN 013-629155-4
$40.00—5/88

PROGRAMMING IN C++
Stephen Dewherst Kathy T. Stark
Prentice Hall
ISBN 013-723156-3
$22.00—5/89

OBJECT ORIENTED ANALYSIS
Coad/Yourdon
Prentice Hall
ISBN 013-629122-8
$29.00—10/89

INSIDE SMALLTALK
Pugh-Lalonde
Prentice Hall
ISBN 013-468414-1
11/89

A FIRST COURSE IN SMALLTALK-80
Hopkins
Prentice Hall
ISBN 013-318387-4
$31.00—11/89

DESIGNING OBJECT-ORIENTED SOFTWARE CONSTRUCTION
RebeccaJ. Wirfs Lauren Wiener Brian
Wilkerson
Prentice Hall
ISBN not yet available 11/89

OBJECT ORIENTED SOFTWARE CUNSTRUCTION 2/E
Bertrand Meyer
Prentice Hall
ISBN 013-629155-4
$40.00—2/90

EIFFEL
Bertrand Meyer
Prentice Hall
ISBN 013-247925-7
$30.00

SEMINAL RESEARCH IN OBJECT ORIENTED DATABASE SYSTEMS
Alfonso Cardenas Dennis McLeod
Prentice Hall
ISBN 013-806340-0
$35.00—1/90

ACTORS: A MODEL OF CONCURRENT COMPUTATION IN DISTRIBUTED SYSTEMS
Gul Agha
MIT Press
ISBN 0-262-01092-5
$25.00—1986

DESIGN & EVALUATION OF A HIGH PERFORMANCE SMALL SYSTEM
David Ungar
MIT Press
ISBN 0-262-21010-6
$30.00—1987

OBJECT-ORIENTED CONCURRENT PROGRAMING
Akinori Yonezawa
MIT Press
ISBN 0-262-24026-2
$32.50—1986

ABCL—AN OBJECT-ORIENTED CONCURRENT SYSTEM
Akinori Yonezawa
MIT Press
$35.00—1990

RESEARCH DIRECTIONS IN OBJECT-ORIENTED PROGRAMING
Bruce Shriver
MIT Press
ISBN 0-262-19264-0
$47.50—1987

CONCURRENT SYSTEMS FOR KNOWLEDGE PROCESSING
Carl Hewitt
MIT Press
ISBN 0-262- 08185-7
$45.00—1989

Product Announcements should be sent to the Editor, HOOT, 8825 SW Umatilla St., Tualatin, OR 97062.

Codecheck 1.0

Abraxas Software, Inc. announced Codecheck 1.0 which aids developers seeking to ensure that their C code is mountable, portable and adheres to in-house style guidelines. The tool is a rule-based expert system that checks C source code for a range of criteria including complexity, formating, and adherance to ANSI standards, including those defined by the user. Portability is another key area that Codecheck inspects, by reviewing code for its capacity to be ported to different C dialects, including C++. Keeping code consistant with standards will help developers port to multiple hardware platforms.

Codecheck is currently priced at $295 but $695 after December 1, 1989.

For more information, contact:
Pat Conley
Abraxas Software
7033 SW Macadam Ave.
Portland, OR 97219
(503) 244-5253

Macintosh® Programmer's Workshop

Apple Computer announced support for C++ with the Macintosh® Programmer's Workshop (MPW™) development environment. The implementation supports Release 2.0 of the C++ language, with extensions to support the Macintosh toolbox and operating systems, Object Pascal based functions and procedures (e.g. MacApp®) and the Standard Apple Numerics Environment (SANE®). C++ source-level debugging is provided by Apple's Symbolic Application Debugging Environment (SADE™). The iostream library, and a SANE-based implementation of the Complex library are included.

MPW C++ also includes an "unmangler" to aid in reading loader messages, and three example C++ programs (two of which are complete Macintosh applications).

MPW C++ requires a Macintosh Plus, Macintosh SE, or Macintosh II with at least 2 MB RAM (4MB highly recommended) and 128K ROM; a hard disk; Macintosh system software 6.02 or later; MPW v3.0 or later; and MPW C v3.0 or later. Availability is planned for early Fall, 1989.

For more information, contact:
APDA
Apple Computer
20525 Mariani Avenue M/S 33G
Cupertino CA 95014
(800) 282-APDA

Smalltalk/V PM Announced

Digitalk, Inc., announced the availability of Smalltalk/V PM, the first programming environment for OS/2 Presentation Manager (PM). This object-oriented development environment is fully integrated with Presentation Manager.

Digitalk also claims that this product is the first and only fully compiled Smalltalk, giving a developer an extremely responsive environment with which to work. The developer has complete access to PM features such as Dynamic Data Exchange (DDE) and Dynamic Link Libraries (DLL) making it easy to integrate and pass data to and from PM tools and other applications. Digitalk sees Smalltalk/V PM significantly easing the burden of developing applications written to run under PM and acting as glue between applications written in other languages.

Smalltalk/V PM is source code compatible with Digitalk's existing Smalltalk/V 286 and Smalltalk/V Mac products allowing portability of applications while preserving the target machine user interface. Smalltalk/V PM preserves the traditional Smalltalk features of source code availability, interactive development, and a large library of pre-existing classes.

Smalltalk/V PM is available now for $499.95 worldwide. An extensive user manual and tutorial is included.

For more information, contact:
Barbara Noparstark
Digitalk, Inc.
9841 Airport Blvd.
Los Angeles, CA 90045
(800) 922-8255 or (213) 645-1082

Smalltalk Apprentice Program

Knowledge Systems Corporation (KSC) announced the initiation of the Smalltalk Apprentice Program, a unique, project-focused training program designed to produce productive Smalltalk programmers in the shortest time possible.

The Smalltalk Apprentice Program is a 3 to 5 week personalized training program designed to expedite mastery of the Smalltalk language and environment.

Participants enter the program with basic skills and a project to be developed. They leave having made substantial progress on the project, often to the point of a functional prototype, and with the advanced skills necessary to continue its development.

The program features:

- Customized, project-focused training
- Small group instructions
- Practical instruction in Object Oriented Analysis and Design
- Training in Rapid Prototyping and Rapid Development techniques
- Instruction in source code change management with a focus on reusability
- All instruction and assistance is provided by expert Smalltalk programmers
- The production of video taped design sessions and project demonstrations
- Access to KSC's Smalltalk products and internal productivity tools

For more information, contact:
Sam S. Adams or Bill Felton
Knowledge Systems Corporation
114 MacKenan Drive, Suite 100
Cary, NC 27511
(919)-481-4000

Sun Unveils Object-Oriented Programming Language

Sun Microsystems has introduced an object-oriented C++ programming language that includes a rich set of integrated programming tools. Sun C++ is one of the first object-oriented language products based on AT&T's recently released C++ Release 2.0.

Sun C++ includes all the new features of AT&T's Release 2.0, such as multiple inheritance. Sun has enhanced its dbx and dbxtool window-based debugger to support object-oriented programming by providing source level debugging capabilities. Many of Sun's other standard profiling and programming tools have been extended to support Sun C++.

Sun C++ 2.0 takes advantage of the advanced features of Sun's UNIX operating system, SunOS, such as dynamic linking and shared libraries. In addition, Sun C++ supports the Network Software Environment (NSE), Sun's configuration management system, which aids in the development of large scale software projects in a distributed environment.

Sun C++, including its integrated programming tools, will be available in October 1989. The cost per tape is $1,200.00.

For more information, contact:
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
(415) 960-1300
Fax (415) 969-9131

Version 2.2 of Eiffel

Interactive Software Engineering Inc. announces the release of version 2.2 of Eiffel, an object-oriented language, method and CASE environment.

Eiffel version 2.2 provides more advanced constructs and tools, including the facility for graphical user interfaces. The new graphical library takes advantage of the most advanced version of the MIT's X Window System. It offers a set of quality pre-defined software components for building applications using windows, pop-up menus, geometrical figures, and mouse and event control.

Eiffel also includes a Basic Library of Data Structures and Algorithms. 2.2 extends the Library with new classes to cover advanced data structures, lexical analysis, parsing, and character-oriented window management.

New tools in version 2.2 include a class browser and new interactive source-level debugging utilities. Also new are a set of advanced language facilities including operator overloading, general purpose copy operations, support of library evolution, more flexible storage allocation, and the introduction of a universal class ANY from which every Eiffel class inherits.

For more information, contact:
Interactive Software Engineering Inc.
270 Storke Road, Suite 7
Goleta, CA 93117
(805) 685-1006
Fax (805) 685-6869

continued from page 1

With this in mind, it is not enough to simply put a check mark next to an object-oriented language simply because it has "some class library". The number of classes available and the quality of those classes should be a major criteria in evaluating an object-oriented language.

For many traditional languages, there exist "code libraries" which can be reused. Due to the procedural nature of these languages, the code libraries are tightly tied to the structure of the data being passed to and from the routines. Programmers may have to write (often complicated) routines which will convert data to and from the required structure. This process reduces the appeal of reusing the code libraries.

If we can produce libraries which are less dependent on the structure of data, the libraries will be much more reusable. Object-oriented languages greatly facilitate this concept since the nature of objects is that their structure is hidden. Although there are situations where knowing the exact structure of data can be of significant advantage (primarily in performance), this knowledge limits reusability.

Smalltalk-80 (from ParcPlace Systems) is the "king of reusability". It is shipped with over 240 classes (based on version 2.3) not including those provided as utilities and examples. While some of these classes are too specialized to be truly reusable and a few are not well designed for reuse, the vast majority of the classes are very solid and can be used time and time again. In addition to these classes, there are a number of reusable class packages commercially sold and supported by Knowledge Systems Corporation and others outside the U.S.. Due to its being a pure object-oriented language, reusable components are greatly encouraged in Smalltalk-80.

Smalltalk/V (from Digitalk) also rates very high in terms of reusability. It is shipped with over 110 classes (based on V286 version 1.1) not including those provided as utilities and examples. There are a handful of classes that are very specialized and will rarely be used in other applications. The classes tend to be fairly solid and reusable with a few exceptions. There are several supported packages available through Digitalk (at various levels of reusability) and Knowledge Systems Corporation. There are also some development tools commercially available from SoftPert Limited.

Objective-C (from Stepstone) comes with a very limited number of classes (about 20). They support various types of collections and are very solid and reusable. A few of the classes, however, are strongly typed and, although very useful, not as generally reusable as their Smalltalk counterparts. Stepstone also sells the ICpak-201, a library of about 60 user interface classes which are generally solid and reusable. Stepstone enthusiastically sells the concept of "Software-ICs", although the commercially available class library for its product is rather limited when compared to that of Smalltalk. As bundled with the NeXT machine, however, the class library is considerably larger.

C++ does not typically come with any class libraries. Some public domain classes are currently available. One vendor, Glockenspiel, has packaged a class library for use in windowing systems. Due to the high level of activity in C++, we expect to see more class libraries becoming available in the near term.

It is important to note that the number of reusable classes available may mean nothing if the classes you need are not numbered among them. Evaluating the class libraries for suitability and quality is a necessary step when weighing a language for reusability.

Development Environment

No matter what domain you are working in, there are going to be classes that do not yet exist. Some code is going to have to be written. The more code needs to be written, the more weight the development environment should carry.

Anybody who has ever used punched cards knows how valuable the "line editor" was to software development. Anybody who has had to use a line editor knows how valuable a full-screen editor is. These differences are less significant than the difference an integrated environment can be over the traditional "edit, save, compile, link, load, test" development cycle in use today.

Dramatic improvements in a software developer's productivity are seen when he can quickly save each segment of code after editing, and re-run the application to test the change without having to go through any extra effort to recompile and relink to the application. Further improvements can be seen during testing if, when an error occurs, the system provides him with the exact spot in the source code where the error was detected, facilitates the identification of the error, allows source code changes to be made, and continue the running of the application. These two facilities are commonly called incremental development and dynamic debugging.

Since we have discussed the value of code reuse, the environment should also make it easier for the developer to browse existing code available for reuse. It is also important that the developer be able to inspect the contents or state of objects in the application.

Both Smalltalks provide rich development environments. They have incremental compilation, dynamic debugging, code browsing, and full inspection features.

Smalltalk-80 has to get the edge as far as the power of the tools included in the environment. Although the compilers, debuggers, browsers, and inspectors of Smalltalk-80 and Smalltalk/V accomplish the same task, there are some handy features and utilities in Smalltalk-80 that do not exist in the Smalltalk/V environment. There are also some code management tools in Smalltalk-80 that are not found in Smalltalk/V.

Additionally, since the Smalltalk environment is highly interactive and graphically oriented, Smalltalk applications tend to provide visual feedback much earlier than applications developed using traditional methods. This can be invaluable when it comes to making changes in the application since the application can be interactively examined at a much earlier point in the development cycle.

Stepstone has an interactive symbolic interpreter and debugger package (VICI) which can be extremely useful in debugging your Objective-C application. However, it does

not keep the developer from having to do all his own file management, compiling, linking, and loading. Other than VICI and traditional tools, there are no other development aids available. Although the interactive debugger provides major productivity improvements over traditional software development methods, it is a far cry from the Smalltalk development environment.

To date, C++ has lacked an integrated development environment. Source code debuggers have been available only from Oregon Software (although others are becoming available). We see changes in this area for C++. ParcPlace Systems has begun shipping ObjectWorks for C++ which brings some of the browsing and incremental development power to C++ as well as source code control. We anticipate a great deal more commercial activity in this area in the future.

Memory Management

Another issue which is tied somewhat to the language and somewhat to the environment is memory management. Various object-oriented languages provide automatic "garbage collection" which occurs in the background while applications are running. This means that developers need not worry about allocating and deallocating memory for objects in the application.

Just as a file system manages the allocation and deallocation of disk space, garbage collection removes the burden of managing memory from the developer's shoulders. It would be a rare scenario for an application developer to justifiably propose using a platform without a file system. We propose that a similar situation exists in memory management for objects.

Smalltalk provides automatic garbage collection as an integral part of the language. Developers only concern themselves with the details of object storage if required by the application. Facilities are available to view and correct the memory management of object.

Objective-C and C++ do not provide automatic garbage collection. The allocation and deallocation of memory for the objects is under control of the developer. In a large and highly interactive application this burden can be significant. I have heard from one developer, at a company which is working at creating a significant class library in C++, that the majority of their QA effort is spent figuring out where and why memory is not being deallocated.

Team Tools

The gains in productivity discussed above focus on the productivity of the individual developer. For successful application development, tools to support teams of developers are required. The hybrid languages can use existing tools to support teams since their development cycle is very similar to existing languages. Since in most pure object-oriented systems the developer can easily change existing code, the management of change using these systems is more critical and more complex. While Smalltalk-80 has some support for team programming and the package from SoftPert supports team development in Smalltalk/V, this area is weak for the Smalltalks.

Performance

Although pure object-oriented languages may be heads above other languages in development productivity, they are not without drawbacks in other areas. These drawbacks tend to be delivery related. As Tom Love (one of the founders of Stepstone) puts it, "Pure object languages consume more computing resources and less people resources than hybrid languages".

Pure object-oriented languages tend to run slower due to dynamic message dispatching. If the application is highly interactive, this is less of a problem since the speed of the user is typically the limiting factor. The difference in performance of the languages is often not significant, but there are applications where every nanosecond counts. It is generally accepted in the industry that C++ has a slight edge over Objective-C, and both of the hybrids have a noticeable edge over the Smalltalks. Recent advances in Smalltalk compiler technology are closing the gap.

Memory Requirements

Pure object-oriented languages tend to take up a significantly larger amount of memory. Much of this memory requirement is due to the integrated environment. As applications grow in size, the impact of this memory requirement is less since it leads to greater reuse which reduces code bulk.

Having the environment so highly integrated in Smalltalk makes it difficult to separate the environment from the application. However, Object Technology International has a manufacturing process which will completely remove the environment from Smalltalk/V applications. ParcPlace Systems and Digitalk will both aid their customers in isolating the application for delivery.

Although the lack of environment is a serious minus for languages such as C++ and Objective-C during development, it is a significant plus when constructing stand alone applications.

Training Investment Necessary to Achieve Productivity

The productivity gains of object-oriented languages does not come for free. A serious investment is necessary to achieve the full potential of object-oriented technology.

Language Training

Training is essential, both formal and on-the-job. Good object-oriented programming will not be successful without good object-oriented design. Good object-oriented design is not possible by designers who are not trained to think in terms of objects. An object-oriented language is not "just another language". There are many who have treated C++ that way and have realized only minor productivity improvements. Ragu Raghavan states that Mentor Graphics is much more successful with C++ because they "don't treat C++ as C+lint". No matter which language you choose, you will

Steinbach & Company

Steinbach & Company is a search firm specializing in the placement of software engineers in the arena of object-oriented programming. Both nationally and internationally recognized, this group provides search and contingency recruitment services for start-up companies and research facilities, as well as product development groups.

Distinctively a software personnel source the company works with developers, architects, managers and executives involved with projects as diversified as object oriented operating systems, compilers, user-interfaces, database, tools and a myriad of other undertakings that are moving forward to meet the future of programming.

Locating appropriate talent, and quality representation is the service Steinbach and Company provides.

For more information, contact:
Dan Steinbach
Steinbach & Co.
Software Personnel Recruiters
ONE Pleasant Street
Maynard, Massachusetts 01754
voice: (508) 897-8661
fax: (508) 897-0677

Taskon A/S

Taskon A/S was established in 1988 as a spin-off from Center for Industrial Research (CIR). Taskon A/S has its technological basis from large national and international projects at CIR, and is the result of a major Norwegian effort within the offshore industry and public services.

Taskon A/S has developed a system integrator, TASKON/INTEGRATOR™, for integrating standard tools (eg. spreadsheets, draw editors) and background services (eg. information databases, calculation programs) into customized task oriented tools. The TASKON/Integrator is the kernel of the TASKON™ Work ENVIRONMENTS which consist of tools directly reflecting the user's tasks and support individual and team work processes. The TASKON Work Environments are flexible and permit changes due to nee requirements from tasks and work processes. It is possible to do rapid design and implementation of new functionality by adding new tools and services, and expanding existing tools and services.

For more information, contact:
Eirik Naess-Ulseth
Taskon A/S
P.O. Box 6
Blindern,0313 Oslo 3
Norway
voice: +47 2 452900
fax: +47 2 452040

The Gunakara Sun Systems, Ltd.

The Gunakara Sun Systems (TGS Systems) is a Halifax-based firm that develops and markets PROGRAPH™, a complete software development system that brings a new level of accessibility to programming the Macintosh.

At the heart of the Prograph system is a completely pictorial, object-oriented, dataflow style language with a built-in source code debugger. The Prograph language and Application Builder are integrated seamlessly, allowing the user to construct, execute and edit high-level user interfaces and their associated methods concurrently in a single development environment.

Prograph's pictorial implementation makes object-oriented programming (OOP) intuitive and easy to learn. It also enables visual animation of methods during execution. Programming power is extended through a utility which allows users to create their own primitives (system defined methods) using C.

For more information, contact:
Peggy Eyre
TGS Systems
The Gunakara Sun Systems, Ltd.
1127 Barrington Street #19
Halifax, NS B3H 2P8
Canada
voice: (902) 429-5642
fax: (902) 429-9983

Lauren Wiener

Writing, Editing,Document Design. Work with a writer who already knows what an object is! I am knowledgeable about object-oriented technology, having spent over two years documenting Smalltalk, applications written in Smalltalk, and writing about various aspects of object-oriented design.

I offer the full range of technical writing services. I will research your subject, conduct interviews, and write new material or edit existing material. I can also design your document, and typeset it using a variety of systems.

In over four years, I have completed a number of projects on time and well, including user and reference manuals, data sheets, research papers and newsletter articles. Writing samples are available.

For more information, contact:
Lauren Wiener
P.O. Box 10351
Portland, OR 97210
(503) 286–4001

The Whitewater Group

The Whitewater Group specializes in applying object-oriented technology to the development of applications for industry standard windowing systems. The Whitewater Group's first product, Actor, is a complete object-oriented development environment for Microsoft Windows. Actor's interactive environment and source code debugger allow you to create standalone Windows programs in less than half the time of the C programming language. Actor costs $495 and runs on any IBM XT/AT PS/2 or compatible machine with 640K.

The Whitewater Group also offers monthly training courses in object-oriented concepts, programming and design. The Whitewater Group provides a range of consulting services including code reviews, and the design and implementation of projects using object-oriented technology.

For more information, contact:
Zack Urlocker
The Whitewater Group
600 Davis St.
Evanston, IL 60201
voice: (312) 328-3800
fax: (312) 491-3905

Oregon Software, Inc.

Oregon Software, Inc., was incorporated in 1977 and over the years has offered standard computer language compilers and cross-compilers for many popular mini and microcomputers. Oregon C++/C, Oregon Module-2, and Pascal-2 compilers are available for Digital, Motorola, Intel, and other processors operating under UNIX, VMS, OS/9, MS-DOS and others. Oregon C++ was the first true commercial C++ compiler available for the Sun-3. Since its introduction in June 1988, we have also released implementations for Xenix/386, and the Sony NEWS and OMRON Luna workstations. A VAX/VMS version and a number of UNIX System V versions will be introduced this fall.

For more information, contact:
JoAnn Bertram,
International Sales Manager
Oregon Software, Inc.
6915 SW Macadam Ave Suite 200
Portland, Or 97219
voice: (503) 245-2202
fax: (503) 245-8449

ParcPlace Systems, Inc.

ParcPlace Systems, Inc., based in Mountain View, CA, is a pioneer and leader in the development and marketing of object-oriented programming systems and graphical user interface technology. ParcPlace products include Objectworks for Smalltalk-80 and Objectworks for C++. Objectworks for Smalltalk-80 is a pure object-oriented programming system which allows developers and programmers to create and deliver graphically interactive software applications. Objectworks for C++, which includes AT&Ts C++ Language System, Release 2.0, is an integrated set of graphically interactive software development tools that allow programmers to create and debug C++ applications with ease and efficiency. In addition to these products, ParcPlace provides a variety of customer training, support, and consulting services. ParcPlace products are supported on a complete range of popular hardware platforms from PCs to high-end workstations, providing portability of graphical applications across diverse processors, operating systems, and windowing systems

For more information, contact:
Helene Tannor
ParcPlace Systems, Inc.
1550 Plymouth Street
Mountain View, CA 94043
voice: (415) 691-6700
fax: (415) 691-6715

Project Technology, Inc.

Recognized for technological innovation and excellence, Project Technology provides training and consulting services in software engineering and project management. Public and corporate courses offered by Project Technology include "Real-Time Object-Oriented Analysis: Information Models", "Object-Oriented Analysis: State and Process Models", and "Information Modeling for Business Applications".

For more information, contact:
Steve Mallor
Project Technology, Inc.
2560 Ninth Street, Suite 214
Berkeley, California 94710
(415) 845-1484

Rational

Rational's product family provides a complete software engineering environment that supports the lifecycle activities of Ada systems development. Rational's technology, which is designed to operate in an open-architecture, heterogeneous engineering environment, supports industry-standard interfaces, allowing automated interconnections with workstations and general-purpose computer systems.

The Rational Environment is a universal host on which software is developed in am automated fashion with a common set of tools independent of the intended execution target. The Environment supports and encourages the use of object-oriented design methods. Facilities for incremental compilation and subsystems help these methods scale up to very large and complex application. Rational products are being used within the largest aerospace and commercial companies throughout the world for the development of a variety of software applications. Many of these applications are being developed using object-oriented design.

Rational headquarters are in Santa Clara, California; European headquarters are in Paris. Regional sales and support operations are in Los Angeles, St. Louis, Washington, D.C., London, and Stockholm, with distributors in Japan,Taiwan, and Australia.

For more information, contact:
Rational
3320 Scott Boulevard
Santa Clara, CA 95054-3197
voice: (408) 496-3600
fax: (408) 496-3636

Servio Logic Development Corporation

GemStone, from Servio Logic Development Corporation, is a complete production quality multiuser object-oriented database management system. It provides all of the features expected in a DBMS including data sharing, data integrity, resiliency, and security.

GemStone's client/server architecture, implemented in C for portability and efficiency, allows maximum flexibility in configuration. Client applications using either Smalltalk or "C" callable interfaces may reside either on the host in a centralized configuration, or on one of the many workstations that GemStone supports in a heterogeneous networked environment. GemStone's expandable hierarchical class structure allows unique or complex data requirements to be easily implemented and maintained. This ability to model complex or irregularly structured data allows GemStone to be used on problems that would be extremely difficult or impossible using any other method.

The GemStone Server is available for VAX/VMS, SUN-3, SUN-4, IBM PC-RT, and DEC 3100.

For more information, contact:
Mark Boyd
Servio Logic Development Corporation
1420 Harbor Bay Parkway
Alameda, California 94501
voice: (415) 748-6200
fax: (415) 748-6227

Software Productivity Solutions, Inc.

Founded in 1984 in Melbourne, Florida, SPS' corporate goals are to research and develop methods and tools which significantly improve software development productivity and software product quality. SPS develops new software and systems engineering technologies through Government-funded studies and prototype developments and through internal research and development programs. SPS applies these technologies by providing consulting services to major Government contractors and commercial software development firms. SPS also applies these technologies in the development of its commercial products.

SPS markets Classic-Ada, a design language and toolset which supports object-oriented development in Ada.

For more information, contact:
Lois W. Valley,
Vice President of Marketing
Software Productivity Solutions, Inc.
122 N. Fourth Avenue
Indialantic, FL 32903
(407) 984-3370

not see truly significant productivity gains and growing class libraries of reusable components until you learn to think "object-oriented".

A common perception is that Smalltalk is much more difficult to learn than the hybrid languages. In general this is not true. The syntax of the Smalltalk language is very simple; incredibly simple when compared to that of C++. However, to be productive in the language, you must learn much more than the syntax. You must become familiar with the development tools, you must learn to think of objects instead of procedures, and you must familiarize yourself with the large reusable class libraries.

Learning what the objects are and becoming familiar with the development tools go hand in hand. The browsers are designed to help you find your way through the objects and class libraries. It usually takes several weeks to get familiar enough with the tools to effectively find objects and classes and several months to understand a significant portion of the classes. Learning the class library is an ongoing process and the more you learn what's there, the more productive you'll become.

If this seems like a long time to learn a language, don't think of it as learning a language – think of it as learning an object-oriented language (e.g. Objective-C), an editor (e.g. 'vi'), a graphics kernel (e.g. GKS), a windowing system (e.g. X), a file interface system (e.g. C's stdio library), numerous development tools, etc. When looked at in this light, the Smalltalks are probably easier to learn than their equivalent traditional counterparts. No matter what language you are learning, becoming familiar with the class library is going to take some time – the larger the class library, the more time it will take. Therefore you will learn C++ much more quickly than you'll learn Smalltalk, but you will have learned much less.

Paradigm Training

Learning how to do "object-oriented thinking" is something which takes time and effort. To quote Tom Love once again, "the greatest hurdle is unlearning current approaches to designing and building software". It is a paradigm shift and is resisted. People tend to hold on to their knowledge and beliefs rather strongly, especially when their knowledge has been tied to their livelihood for so many years. This can be overcome through training, time, and experience.

Although a pure object-oriented language with a large class library may take longer to learn than a hybrid language without a large class library, it will greatly enhance the learning of "object-oriented thinking". The uniformity of the usage of objects and the many examples of objects used in the system help get people familiar with what objects are, how they work, and how to create new ones.

Languages which are hybrids do not force the use of the object-oriented thinking. By allowing procedural code to be written, the developer may take more time to understand and take advantage of the power of the technology. Using a object-oriented language to develop procedural code will leave the developer feeling that there is no advantage to be gained. Since there are few classes included as examples to either study or use with C++, it has been fairly common for this attitude to develop in new users of C++.

Reusable components are extremely valuable, but are not automatically created just because one uses an object-oriented language. They must be carefully engineered and maintain the integrity of object-oriented concepts in order to be widely reused. Designing and implementing reusable components is a skill that takes both training and time.

Other Issues

Although productivity is a major selling point for object-oriented languages, and is worth a significant investment, it does not overcome all requirements. There are many drawbacks of current object-oriented technology when it comes to dealing with delivery of applications.

Delivery Platform

If an application has to run on the "Batmobile 750", you may not have a large variety of object oriented languages to choose from. The range of platforms supported by the various object-oriented language vendors is limited when compared to those supported by traditional language vendors.

Smalltalk/V is currently available on the Macintosh and the PC. Smalltalk-80 is available for the 386 PC, the Macintosh, and most major UNIX workstations. Object Technology International of Ottawa has ported Smalltalk to some embedded systems.

C++ is starting to become available on a wide range of platforms from several vendors with some compatibly problems. Objective-C is commercially available on half a dozen different platforms, although ICpak-201 is not available for the PC.

Obviously, it is important to find out the support available for the particular platform(s) of interest before taking any steps. The availability of the various languages is changing rapidly and vendors should be contacted for the latest information.

Utilizing Existing Software

An application may need to utilize existing code written in other languages. The advantages of hybrid languages in this regard is that they are typically compatible with code written in the base language (e.g C++ is totally compatible with C). In these cases, integration with existing code in the base language is trivial.

Smalltalk-80 and /V both offer mechanisms by which interaction can be made with existing programs and/or extended with "user-defined primitives" written in another language, typically assembler or C. Smalltalk-80 on the UNIX platforms allow for higher level interprogram communication such as sockets or pipes.

User Interface Support

There may be unalterable user-interface requirements ("look and feel", or total integration with other applications and/or windowing systems).

Smalltalk-80 has its own windowing system which can be a problem if integration with a particular UI system is nec-

essary. If common look and feel is desired, the windowing system in Smalltalk can be modified to reflect the look and feel desired, (Not recommended for inexperienced Smalltalkers!).

Although an integrated windowing system which is different from the host windowing system appears to be a negative, their are applications where the same look and feel is desired no matter on which platform the application is running. Here Smalltalk-80 is a major win as it is 100% portable from platform to platform. On many platforms you can run Smalltalk inside a host window, but it uses its own windowing system inside of that window. One drawback of Smalltalk-80 is that it currently has little support for color.

Digitalk has taken a slightly different approach with Smalltalk/V. On the Macintosh, you get Macintosh windows. The current version of Smalltalk/V on the PC has its own integrated windowing system that appears similar to Microsoft Windows. The recently released Smalltalk V/PM uses Presentation Manager as the underlying windowing system. They've adopted the approach that windows produced by Smalltalk should look like all the rest of the windows on the system. Applications written in one of these systems will port to the others with a minimal amount of effort.

Objective-C's ICpak-201 provides a windowing environment which is its own although it, much like Smalltalk-80, can have its windows live inside a host window. Since it is also "C-compatible" Objective-C can interface to any other windowing system which has a C interface.

C++ supplies no windowing system. It, too, is C-compatible. In addition, many of the X tool-kits which are now available provide significant support for C++ development. CommonView is an example.

Summary

The productivity gains possible through the use of object oriented languages is impressive. Significant increases in productivity come mainly through reusability and the development environment... don't underestimate the importance of these two issues. Also don't overlook the investment necessary to enjoy the benefits.

Before writing off a language because it is missing a checkmark, determine the significance of that checkmark in your application area. If it is significant, ask yourself if enough resources can be saved due to the other features of the language to overcome the missing checkmark.

None of the languages are a panacea. Each comes with its own set of advantages and shortcomings. It is important to realize how those features will affect your specific application area in both the short and long term.

Ken Auer, *Senior Member of the Technical Staff at Knowledge Systems Corp. Ken started working with OO Technology over four years ago at Paradyne Corporation where he led several development efforts.*

Analysis and Design

Understanding Object-Oriented Analysis

by Sally Shlaer & Stephen J. Mellor

Soon after you started investigating CASE tools and methods for your software development project you probably discovered that there are a huge number of options to choose from. The distinctions between these options often seem very subtle—especially in the area of structured analysis. What are the differences? Do they really matter?

We think they do and so have assembled information about object-oriented analysis, the newest member of the structured analysis family. In addition, we offer a few pointers that should help you tell object-oriented analysis from some of its cousins and discuss the application of this method in the practical world.

Goals and Philosophy

Object-oriented analysis aims to describe the operation of a part of the real world. To be effective, such a description must provide a global view that can be examined for coherence, completeness, and consistency. In addition, the description must explain the details of the operation in such a way that each detail can be examined and definitively confirmed or refuted.

To meet these goals the object-oriented analyst constructs three types of formal models: an information model, a set of state models, and a set of process models. Models of these types have been used in software engineering for some time, but in object-oriented analysis all three types of models are used together and in a highly constrained and simplified manner.

In developing the object-oriented approach we have selected a minimal set of well-factored constructs for the modeling language to reduce the number of notations and forms the analyst must use in constructing the models. The net effect of the approach is one of simplicity and a certain mathematical elegance.

When to Use Object-Oriented Analysis

Object-oriented analysis has been applied on about 50 software development projects since 1979. While the method is applicable to a very wide range of problems, we've found it to be especially effective in these situations:

Large Projects. It's practical to have several teams working in parallel on different parts of a project. Because data

Reprinted with the permission of Hewlett-Packard Co.'s Design Center Magazine, November 3, 1988.

into a comprehensive approach to modeling software architecture.

IDE, founded in 1983, develops and markets the Software through Pictures integrated CASE environment. Software through Pictures offers a full-lifecycle solution; its open architecture permits users to tailor the tools to meet their individual requirements. It supports several different approaches to real-time requirements specification, a list that will soon include OOSD (available 10/89 or sooner). Software through Pictures includes applications for automatic documentation preparation, DOD-STD-2167, and 2167A, and requirements traceability. It also generates data declarations for Ada, C and Pascal, as well as SQL schemas for popular relational database systems. Software through Pictures runs on Digital's entire VMS and ULTRIX VAXstation and RISC-based DECstation workstation family and on all other major UNIX workstations, at prices ranging from $5000 to $21,000.

For more information, contact:
Nobby Akiha, Director
Marketing Communications
Interactive Development
Environments, Inc., (IDE)
595 Market St., 12 th Floor
San Francisco, CA 94105
(415) 543-0900

Knowledge Garden Inc.

Knowledge Garden is a company with more than technical excellence - it has a guiding philosophy.

Our work reflects our belief that the computer can be used as a tool to enhance the expressive and artistic abilities of the individual.

Knowledge Garden sees the "displayed word" as a new communication medium. Products are directed towards putting this new medium and the tools to use it, in the hands of communicators, whether or not they possess any special technical background.

Bev and Bill Thompson, technical directors of Knowledge Garden, have spend the last four years designing expert system tools and applications. They are well known authors with columns and articles published in AI Expert, Byte, Computer Language, and Dr. Dobbs magazines.

The Thompsons first commercial software product was the experts system shell MicroExpert (MasterExpert in Europe). MicroExpert which was published by McGraw Hill in July, 1985, and has been translated into Italian.

In December, 1986, the Thompsons joined forces with John Slade as president to found Knowledge Garden, Inc. Slade has worked at all levels of the computer industry, most recently as co-founder of Blackbaud MicroSystems, Inc. Having sold his interest in Blackbaud, Slade comes to "the Garden" as an experienced manager with a clear understanding of the link than must be made between a new technology and its practical application.

For more information, contact:
Debra Lee Pellerin
Knowledge Garden, Inc.
473A Malden Bridge Rd.
Nassau, NY 12123
voice: (518) 766-3000
fax: (518) 766-3003

Knowledge Systems Corporation

Knowledge Systems Corporation (KSC) provides a full range of products and services designed to assist its clients in the cost-effective application of object-oriented programming technologies.

The Smalltalk Apprentice Program is a personalized, project-focused training program designed to develop productive Smalltalk programmers in the shortest time possible.

The Rapid Prototyping Center provides KSC's clients with access to object-oriented programming technologies aimed at reducing software life cycle cost through the development of functional application prototypes. KSC combines its expertise and access to a large volume of reusable Smalltalk components, to provide the most productive prototyping environment available.

The Smalltalk Collection is a library of reusable Smalltalk tools and applications. Current offerings include: Draw-80, NodeGraph-80, PluggableGauges, UserInterfaceTools1, and MathPak1.

Smalltalk Applications Center: KSC specializes in Smalltalk, a pure object-oriented development environment, ideal both for developing prototypes and for implementing final systems based on object-oriented designs. KSC provides development services to government agencies and Fortune 500 corporations.

For more information, contact:
Nancy Norfleet
Knowledge Systems Corporation
114 MacKenan Drive, Suite 100
Cary, NC 27511-6446
(919) 481-4000

Object Design Inc.

Object Design, Inc. was established to develop and market object-oriented database management software. The company's objective is to bring products to market that will be the standard for engineering design, computer-aided publishing and computer-aided software engineering applications. The company is positioned to transition aggressively into the commercial DBMS market as object-oriented technology begins to make its presence felt in the next two to three years.

The company's initial product, ObjectStore, is designed to serve as substrate for large-scale complex applications requiring high performance and the ability to handle non-record oriented data. Users of ObjectStore can expect to gain competitive advantage through a significant reduction in both the cost and time required to develop database-intensive applications. Applications. built using ObjectStore will be significantly less costly to enhance and maintain.

For more information, contact:
Eugene A. Bonte,
Vice President/Marketing
Object Design Inc.
One New England Executive Park
Burlington, MA 01803
voice: (617) 270-9797
fax: (617) 270-3509

Object Technology International Inc.

Object Technology International Inc. (OTI) is an advanced software engineering company. OTI provides tools, consulting and training to enable the development, delivery and support of object-made systems and products. OTI specializes in embedded systems and turnkey applications development.

OTI is a world leader in the development of embedded applications using object-oriented technology. Typical applications range from dedicated special purpose controllers to multiprocessor VME and FDDI based distributed systems. OTI Smalltalk+C tools allow embedded systems to be designed, developed, delivered and maintained using a single environment.

OTI's turnkey applications group provides complete custom solutions built to customer requirements. Our engineers think of the application first and then apply the OO technology where and when it is appropriate. OTI's software engineers become part of your software engineering team dedicated to completing your application or product. OTI supports the entire life cycle from architecture definition to integration and test.

For more information, contact:
Dave Thomas
Object Technology International Inc.
1785 Woodward Drive
Ottawa, Ontario
Canada K2C 0P9
voice: (613) 228-3535
fax: (613) 228-3532

In 1986, Smalltalk/V, a graphical version for the IBM PC, was introduced. In 1988, Digitalk introduced versions of the Smalltalk environment for 286/386 computers and the Macintosh. In October, 1989, Digital introduced Smalltalk/V PM for OS/2 Presentation Manager. The company also offers training courses and a set of reusable software components. Digitalk products are available through a network of computer and software dealers in the U.S., Europe, and Asia, or directly from Digitalk.

For more information, contact:
Barbara Noparstak
Digitalk, Inc.
9841 Airport Blvd
Los Angeles, CA 90045
voice: (213) 645-1082
fax: (213) 645-1306

GoldHill Computers, Inc.

GoldHill is a leading supplier of expert system programming tools for advanced software applications. GoldHill offers products, training courses, and consulting services to help customers develop and deliver expert systems on desktop computers. Founded in 1984, GoldHill has played a substantial role in bringing PC-based expert system technology into mainstream corporate computing.

GoldHill's product line includes a broad range of tools for training as well as application development and delivery. Our flagship product, GoldWorks II, is the first PC-based expert system shell to combine rule-based processing with frames and a robust implementation of object-oriented programming.

GoldHill is a charter member of the recently-founded Object Management Group (OMG), an international consortium formed to promote the adoption and implementation of a common object-oriented applications environment. Based on its experience in developing object-oriented tools like GoldWorks, the company expect to make significant technical contributions to OMG.

For more information, contact:
Ames Abbot
GoldHill Computers, Inc.
26 Landsdowne Street
Cambridge, MA 02139
(617) 621-3300

Graphael, Inc.

Graphael, Inc. specializes in object-oriented database management software and application development. Its product line is successfully delivering solutions to industrial applications, and includes: G-BASE, the first object-oriented database management system commercially available; G-LOGIS, a logic-programming language for developing knowledge-based applications; and tools to complement the database environment, including GREG, a graphical display tool for database structure; G-ACCESS, a menu-based query interface; PRO-QUERY, a PROLOG-like query language; and G-REPORT, a report generator. In addition to its products, the company offers broad technical support for service and installation, and knowledge engineering and application consulting services. Graphael, Inc. is headquartered in Waltham, Massachusetts, with sales offices in Washington D.C., Chicago, Illinois, and Mountain View, California.

For more information, contact:
Joe Hancock, Vice President, Marketing
Graphael, Inc.
255 Bear Hill Rd.
Waltham, MA 02154
(617) 890-7055

Instantiations, Inc.

Instantiations specializes in making object-oriented technology practical for its clients. The company uses its 5 decades of cumulative experience in the field of object-oriented technology to make available a full range of services including:training, consulting, software development, R&D, product evaluation and support, and market development. Training courses include: Object-Oriented Design, Beginning Smalltalk, Building Applications in Smalltalk. Custom, tutorial, and on-site training is available

For more information, contact:
Mike Taylor
Instantiations, Inc.
921 SW Washington
Portland, Oregon 97205
voice: (503) 242-0725
fax: (503) 242-0729

Interactive Software Engineering, Inc.

Interactive Software Engineering was founded in 1985 with the aim of improving software quality in industrial environments through tools, methods education and consulting. The company developed Eiffel, the most advanced language, method, and CASE environment available, which adds the full power of Object Oriented programming to modern software engineering principles. Its conception emphasizes reusability, extendibility, correctness and robustness. The language supports multiple inheritance, dynamic binding, static typing, assertions, exceptions, etc. The environment offers automatic recompilation, documentation, garbage collection, debugging, graphical design. Cross-development from Eiffel to C is supported, allowing the production of stand-alone C packages. Other tools marketed by Interactive with Eiffel technology include a language-independent syntax-directed editor and window manager. ISE is also involved in training and consulting on software engineering methods and tools and offers worldwide courses on object-oriented design and programming as well as other advanced software engineering topics.

For more information, contact:
Thomas P. McCarthy
Interactive Software Engineering Inc.
270 Storke Road, Suite 7
Goleta, CA 93117
voice: (805) 685-1006
fax: (805) 685-6869

Intellipro International Inc.

Intellipro International Inc. is a New Jersey-based computer software company specializing in the development of advanced knowledge engineering environments for engineers and scientists. OPS-2000, Intellipro's primary knowledge engineering product, made Intellipro the first company in the world to design a knowledge engineering environment for the C++ programming language.

Intellipro's technology continues to impact the state-of-the-art in both expert-systems and object-oriented technologies. Intellipro was the first computer software company to make a single product available for MS-DOS, Microsoft Windows, OS/2, and any UNIX-based computer architecture, which represents a significant commitment by Intellipro to handle the diverse needs of its customers.

For more information, contact:
Frank Lopez
Intellipro International Inc.
10 Park Place, Suite 224
Morristown, New Jersey 07960
(800) 4-INTELLIPRO

Interactive Development Environments, Inc.

Interactive Development Environments, Inc. (IDE) recently announced publication of a non-proprietary notation for design of software systems called Object-Oriented Structured Design (OOSD). OOSD is the first general object-oriented approach; it is language-independent, where previous approaches each have been specific to a programming language. It is built upon ideas and notation from traditional Structured Design and Booch's Object-Oriented Design, among others. OOSD synthesizes traditional top-down design with modern concepts of object-oriented design

and processes are so highly localized to the objects, the different teams rarely get in one another's way.

Changing Requirements. The models are quite stable: it's easy to add an object, attribute, or event without significant ripple effects. In addition, because the models are so complete, they make it easy to anticipate exactly where the requirements are most likely to change.

Weak Requirements. When requirements are initially vague, incomplete, or inconsistent, the method is very helpful in identifying and articulating the requirements.

New Applications. If the application is new to the software developers, the method provides a systematic approach for intensive investigation and understanding.

Diverse Expertise. A project involving diverse areas of expertise (e.g., engineering, operations, finance, etc.) often suffers from the lack of a standard vocabulary. The information model provides a highly effective, readily learned lingua franca for project participants.

Perfect Logic. Some systems must perform with perfect logic lest they endanger expensive equipment or human life. Examples of this are cancer therapy machines and embedded microprocessor systems in automobiles or aircraft. For these projects, state analysis works well to allow investigation of the results of sensor or actuator failure and to detect logic problems.

STEP ONE
The Information Model

The cornerstone of object-oriented analysis is the information model, which is used to identify the conceptual entities of the world we are modeling. The information model formalizes knowledge about the world in terms of objects, attributes, and relationships. The object-oriented analyst looks to three definitions for guidance in making the required conceptualization and formalization:

Object: an abstraction of a set of real-world things such that all of the things in the set — known as instances — have the same characteristics and are subject to and conform to the same rules.

Attribute: the abstraction of a single characteristic possessed by all the instances that are, themselves, abstracted as objects.

Relationship: any pattern of association that holds systematically between two or more sets of real-world things abstracted as objects.

Forms of the Information Model

The information model is produced in two forms. One form consists of a set of textual definitions for each object, attribute, and relationship describing the basis of abstraction. These are subject to detailed, intensive review.

A second, graphical form provides a global view. This form supports examination of the model for coherence, consistency, and completeness.

Instances

To understand a real-world problem at the level of instances, it is helpful to think of each object in the information model as a table. Each instance of an object, then, is represented by the data in a single row of that table.

The tables do not actually appear in any of the formal models of object-oriented analysis, but they are a useful mental image for the analyst working out complex constructions. Many constructions that initially seem plausible in the graphical form can be tested and rejected by constructing the implied tables and trying to fill in sample instance data.

STEP TWO
Lifecycles

Once the information model has been produced, we turn our attention to the dynamic behavior of the conceptual entities and associations.

The object-oriented analyst assumes that all objects and relationships have lifecycles. A lifecycle is composed of a number of stages; during any particular stage, specified physical laws, operating policies, rules, and the like regulate the behavior of an instance of the object. The lifecycle may be interesting. For instance, a car has the following lifecycle stages:

1. Stationary with engine off;
2. Stationary with engine on;
3. Engine on going forwards;
4. Engine on going backwards;
5. Engine off going forwards;
6. Engine off going backwards; and
7. Car destroyed.

Some objects have trivial lifecyles, such as the Department of Motor Vehicles:

1. Exists.

The important thing about lifecycles is that an instance can follow only certain patterns in progressing from one stage to another. A car cannot go directly from stationary with engine off to engine on going forwards; it will have to go through at least one intermediate state.

In object-oriented analysis, a lifecycle can be formalized for each object and relationship in the information model. Lifecycles are formalized in state models, using the following notions for guidance:

States: a condition of the object during which a defined set of rules, laws, and policies apply.

Events: real world occurrences that signal us when an instance of an object is moving from one stage of its lifecycle to another. In the car example, a car goes from stationary with engine on to stationary with engine off when the engine dies. Such an occurrence has two aspects: when it happens and what it happened to (which car). These two aspects are formalized together as an event: a control signal that carries with it the identifier of the object or relationship that is affected by the occurrence.

Actions: a set of processes that take place immediately when an instance of an object (or relationship) arrives in a state. The action processes can write attributes of the object or relationship with which they are associated, and they can make queries of objects and relationships via access paths along the relationships on the information model. An action process can also generate events and set timers. Finally, every object or relationship that has a state model must also have a status attribute (i.e., car.status) to serve as a state variable. So for each state, there must be an action process that updates the status attribute to record which state was most recently entered.

Coordinating Lifecycles

Few objects actually go through their lifecyles independently. In any real problem, the dynamic behavior of the instances is highly correlated through physical laws, operating policies, and the like. Some examples might include:

- Two airplanes had better not be in the same place at the same time.
- An energized cable and a de-energized cable cannot be connected by a closed circuit breaker.
- A clerk is allowed to serve only one customer at a time.

These kinds of restrictions, coordinations, and synchronizations are formalized through the state models, making use of the fact that an action process in one state model can generate an event that is input to another state model.

Forms of the State Model

The state models are documented in several forms. To provide the required level of detail in the documentation, the state model for a single object or relationship can be described in the familiar state transition diagram. The state transition table is an alternative representation of a state model. In practice, we find that both are usually required: the diagrammatic form helps you to understand the fundamental flow of the object's lifecycle, but the tabular form is required for careful, exhaustive verification.

At the global system level, the interactions between the state models can be readily seen in a graphical representation called an object communication model. This diagram shows each state model as a box and communicating events and their data as connecting arcs.

Finally, as an aid to the analyst during model construction, as well as an aid in review, we find it helpful to maintain an event list. In formal documentation, events are given identifiers (numbers such as E34), meanings (e.g., boiling point achieved), and required data content (the name of one or more attributes from the information model). The event list is simply a tabulation of this information.

Lifecycles of Instances

The state model represents the behavior of an object or relationship. To represent the behavior of instances of the object or relationship, you can think of the state model as a rubber stamp: from it you can produce a separate state machine for each instance.

Tokens can be placed on the instance state machines to simulate or track the behavior of the lifecycles of instances. Again, the separate state machines for instances do not appear in the formal documentation, but are a useful mental tool for the analyst.

STEP THREE

Process Models

The third step is the construction of a set of process models. In this step a separate data flow diagram is constructed for each state in each state model. The data flow diagram for a state graphically depicts the action processes associated with that state. This is a fairly mechanical step, since the content of the action processed is defined as part of a previous step.

The simple DeMarco form of data flow diagrams is used, with the following additional simplifications:

1. With the exception of a few system-wide pieces of data (time of day, date, and similar universal data), only attributes from the information model are allowed to appear on the data flows.
2. The only data stores allowed are those that correspond to the tables implied by the information model. These can be considered to be true data stores in that the relevant data persists from state to state.
3. The data stores are shared, as needed, by all data flow diagrams in the system.
4. Since the action processes have the nature of primitive processes or queries, no further partitioning or decomposition of the action processes is needed (or allowed).
5. The same action process can appear on multiple data flow diagrams if the process is to be executed as the result of arrival in multiple states.

Forms of the Process Models

The process models are documented in two forms. At the detail level are the data flow diagrams, organized so that all data flow diagrams for a single state model are together. No process descriptions are provided, since that purpose is fulfilled by the action processes in a previous step. A global view of the processes is provided by the object communication model described above.

Value of the Process Models

The value of the process models in object-oriented analysis is considerably less than in other more function-oriented approaches because the process models add essentially no new information to the analysis. However, building the process models does allow the analyst to make an important refinement: by comparing the data flow diagrams for all the states of a single state model, you often find that the same process — or some very similar processes — are being used over and over again. In this case, it is advantageous to make slight adjustments in the definitions of the action processes to maximize their reusability.

Sally Shlaer & Stephen J. Mellor, *principles at Project Technology, Inc. Project Technology offers tranining and consulting services in software engineering and project management.*

Company Showcase: An Information Service

Regularly appearing in HOOT, this section introduces you to companies offering various products and services in the object-oriented field. Every company which sells an object-oriented product or service (old or new) will eventually appear in the Company Showcase.

Included here is a brief description of the company and its product or service, atributes, with name and phone listing for further contact. This section is provided as an information service to readers and no endorsements by HOOT are implied. Summaries are provided by the companies. Companies wishing to be included in the Company Showcase should send appropriate information to Jeff McKenna, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062. Tel. (503) 692-1889

Apple Computer, Inc.

Since 1983, Apple Computer has been a leader and pioneer in the commercial use of Object-Oriented Programming. As a result of our early and timely commitment to OOP, Apple's rich history includes implementations of Smalltalk, Clascal, the Lisa Computer Toolkit, Object Pascal, MacApp, HyperCard, Macintosh Allegro Common Lisp, and C++. Object Pascal, the Pascal extensions that let the language handle objects, was created in 1985 through the collaboration of Apple and Nikolas Wirth (creator of Pascal). Apple has also selected C++ as a key programming language for use in writing our distinctive graphics-oriented Macintosh operating system. Apple's unwavering endorsement of OOP resulted in the first wave of commercially available software products created using OOP techniques. Using MacApp, Apple's third-generation OOP framework, developers are currently delivering commercial solutions in a wide variety of areas such as accounting, CAD, OCR, word processing, report generation, scientific visualization, and mapping. In addition, at our recent Worldwide Developer's Conference, over 1,000 enthusiastic Apple developers attended special seminars on C++, MacApp, and OOP. We believe that recent public affirmations of OOP further validate the pioneering efforts begun by Apple in the early 1980's. OOP is the wave of the future and has now become a mainstream industry imperative.

For more information, contact:
Ron Lichty, Manager, Object Oriented Platforms Product Marketing
Apple Computer, Inc., M/S 75-6B
20525 Mariani Avenue
Cupertino, CA 95014
voice: (408) 974-5576
fax: (408) 974-9456

AT&T

AT&T maximizes the benefits of quality products through quality training. We provide our customers with first-class training, providing them with the skills and knowledge to get the most out of AT&T products. Fundamental through advanced level courses are offered in UNIX System V, C and C++ languages, Computer Technologies, Networking and Data Communications.

For more information, contact:
AT&T, Education
and Training Registration
55 Corporate Drive Room 13C66
Bridgewater, NJ 08807-9903
(800) 554-6400

Borland International, Inc.

Borland International, Inc. is a leading developer of high-performance microcomputer software products. Borland's Professional language Series includes critically acclaimed developer's tools such as Turbo Pascal and Turbo Assembler &Debugger. Borland offers in its Professional Business Series some of the world's most widely used business applications and developer's tools such as Paradox, Quattro, SideKick Plus, Reflex, Sprint and SideKick for Presentation Manager—-the first ever application for IBM's OS/2 Presentation Manager. The Scotts Valley, California—based company, founded in 1983, ranks among the leading U.S. microcomputer companies.

For more information, contact:
Tami Casey (408) 439-1619
Borland International, Inc.
1800 Green Hills Road
Scotts Valley, CA 95066
voice: (408) 438-8400
fax: (408) 439-9273

Complete Computer Corporation

Complete Computer Corporation of New York has developed Complete CTM, the Object-Oriented Toolkit for C, which combines object-oriented programming with powerful utilities designed to increase your productivity by 300%. First among these is the Complete C Driver which contains an application dictionary to automatically monitor all of your classes, methods, and dependencies. This utility replaces the "make" file, and becomes a continually updated central reference for every stage of development.

Complete C comes equipped with an object-oriented logical debugger allowing the programmer to walk through the object hierarchy, trace messages, and display objects. Then, when you're ready for release, the Application Streamliner will strip out unused code from the application to produce the smallest possible final executable, and Complete C will optimize your code to use static binding while doing strong type checking. Complete C also ships with a documentation generator which outputs directly from your source code, as well as 400k of powerful, easy to use foundation classes with source code which will get you up and running in no time.

For more information, contact:
David Goodman,
Marketing Vice-President
Complete Computer Corporation
111 West 57th Street Suite 1400
New York, NY 10019
voice: (212) 582-2635
fax: (212) 397-1580
bbs: (212) 956-4535

Corporate Computer Systems, Inc.

Corporate Computer Systems is comprised of three organizations which include Software products for HP 1000/3000/9000 systems, Custom Hardware and Software Systems to Fortune 500-sized companies, and the CCS Consultants personnel service which provides both permanent and contract professionals, primarily to the telecommunications industry.

Among CCS' best known software products is CCS/C++ for the HP9000 system.

CCS/C++, a licensed port of AT&T release 1.2, improves on the C programming language by adding features to allow object-oriented programming, function argument defaults, user defined types, operator overloading, complex arithmetic, and much more. Relocatable Binary License is $1,000 for a 16-User System; $2,000 for a 32-User System; and $3,000 for a 64-User System. Contact CCS to request a demonstration agreement to receive your free trial copy.

For more information, contact:
Ms. Joan Dillon,
Director, Sales/Marketing
Corporate Computer Systems, Inc.
33 West Main Street
Holmdel, N.J. 07733
voice: (201) 946-3800
fax: (201) 946-7167

Digitalk, Inc.

Digitalk, Inc., headquartered in Los Angeles, CA, was founded in 1983 to develop and market object-oriented programming tools for personal computers. The company pioneered the introduction of Smalltalk to the personal computer world in 1985 with Methods, the character-based Smalltalk implementation for the IBM PC.

Object-Oriented Systems: The Commercial Benefits

by Judith Jeffcoate, Keith Hales and Valerie Downes *(Ovum, Ltd, 1989)*

One of the more interesting questions surrounding object-oriented technology is: Is there a Object-Oriented Market? This report by Ovum does a reasonable job of addressing that question.

Ovum offers a range of marketing reports in computer related areas and this report acknowledges the importance of objected-oriented technology in that marketplace. It is certainly the most comprehensive such effort that this reviewer has seen to date with over 400 pages. Because Ovum is based in England, the U.S. reader will find much more information about European companies than is normally available.

The report starts with a extensive management 'summary' (some 23 pages) that does a reasonable but uneven job of presenting the essence of the technology and identifies the major trends.

The next section describes the concepts and application of OO technology in some depth with a good number of examples from a variety of sources.

The third section identifies a wide range of products and describes many in depth.

A market analysis section follows with world-wide projections into 1995. The primary source for this data was obtained from the individual vendors.

Detailed supplier profiles from some 24 companies are next followed by an an interesting section of seven case studies of OO users. A company profile section of some 60 vendors completes the report.

Overall, the report is impressive in the amount of data presented. It is weak in making some sense out of that data. The market segments proposed by the authors do not match well with the activities of the players suggesting that some alternative segmentation might work better.

Although the published date was April,1989, the report appears older than that. A few technical details were out of date by some two years and details of some company profiles were noticeably out of date. This might be expected in a field that is evolving quickly.

The Ovum report ties OO activities tightly to AI activity and makes the correct observation that CASE vendors will be moving strongly into the OO area..It does not address the reusable software market, which is really the only new market which is likely to emerge from the technology. The lessons from the recent history in AI should help us see that a technology does not necessarily a market make.

It was difficult to determine where the market projection figures came from other than the usually optimistic, hockey-stick projections for new start ups. The market impact of OO technology merging into mainstream computing was not addressed sufficiently.

Regardless of these deficiencies, the report does give a broad overview from the marketing point of view and contains a good deal of data. It has attempted to present a rather complex and immature field in a summary manner. It is the best effort to date tackling this problem. It receives a B-.

Reviewed by
Jeff McKenna
Editor, HOOT

Object-Oriented Programming: Myths and Realities

by Zack Urlocker

As with any new, promising technology, there's much confusion about object-oriented programming among managers and programmers alike. At present, there's no universally agreed upon definition of the terms for object-oriented programming among experts. How can we expect newcomers to fully understand the benefits? The object-oriented world seems to be expanding so rapidly that the words "object-oriented" have become synonymous with "good" in the eyes of many.

Since this is the first HOOT article on object-oriented environments and languages, I'm going to begin by looking at some of the myths of object-oriented programming (OOP) to help pinpoint what it really is and isn't. In future articles I'll discuss the relationship of OOP languages and programming environments, graphical user interfaces and application frameworks.

The Myths of Object-Oriented Programming

The real meaning of object-oriented programming has been lost or at least misplaced. Let's take a look at some of the myths that prevail and try to clear up the confusion.

MYTH: *Object-oriented programming is a single language.*

Many professionals are aware of only a single object-oriented language, be it Smalltalk, C++, Eiffel, Actor or some other language. As a result, they confuse the concepts of object-oriented programming with the language implementation. This sometimes results in a dismissal of the technology based on the details of a single language.

For example, most object-oriented languages, including Smalltalk, Actor and Objective-C use single-inheritance, where each class has a single ancestor or super class. Other languages, such as Eiffel and C++ 2.0 allow multiple inheritance, where behavior can be inherited from more than one ancestor class.

There are many similar concepts in object-oriented programming which are implemented differently in particular languages. If managers understand the broad concepts of object-oriented programming they are better able to evaluate languages to determine which will best meet their needs.

MYTH: *Object-oriented programming is graphical.*

Since some object-oriented languages operate in a graphical programming environment, many people have incorrectly assumed that object-oriented programming was only applicable to graphical applications. Certainly the integrated graphical environments in Smalltalk and Actor are strengths of these products. Other languages, such as C++ and Object Pascal, are traditional standalone compilers without graphical features.

Although object-oriented programming can be applied effectively to graphical and windowing applications, it can also be applied to non-graphical systems including real-time control systems, operating systems, embedded systems and traditional character based applications.

The fact is, object-oriented programming is a paradigm that can be used for almost any type of application, as long as you pick the right tools.

MYTH: *Object-oriented programming is inefficient.*

Many early implementations of object-oriented systems were extremely demanding of computer resources and ran best on dedicated hardware. Since these early systems were used for research and were not suitable for production development there is a perception that object-oriented programming is inherently inefficient.

However, as a result of improvements in language translation and code optimization, today's object-oriented systems are far more efficient than before and are being used successfully for a variety of performance-critical systems.

There is nothing inherently inefficient about object-oriented programming. Many newer object-oriented languages give the programmer the option of using techniques such as early binding and as a result, execute as fast as procedural languages.

MYTH: *Object-oriented programming requires abandoning existing tools.*

In many cases, professional programmers and managers are unaware of the ability of object-oriented languages to coexist with other tools and languages. Professionals do not have to abandon existing tools and libraries in order to take advantage of object-oriented programming. Modern object-

oriented languages allow you to call routines written in procedural languages, make use of operating system calls, read and write existing data files and so on.

The ability to coexist with other languages and tools means that organizations can convert to object-oriented technology in a way that matches their own requirements. The investment in non-object-oriented development technologies is too high to ignore.

MYTH: *Object-oriented programming is difficult to learn.*

Because the original developers of object-oriented languages invented new terminology and in some cases new language syntaxes to express OOP concepts, there is a perception that object-oriented programming is difficult to learn. Programmers with many years of procedural programming experience feel lost with these new ideas.

Although it is probably true that you can teach a neophyte object-oriented programming faster than you can teach a 15 year COBOL veteran, any programmer can learn OOP with the right perspective and support. In my experience teaching OOP, I've found that the concepts are quite easily understood, but their application requires some "unlearning" on the part of some experienced programmers. It is interesting to note that many advanced programmers find that the have been trying to incorporate OOP concepts into their own procedural code for years. When they are given a language that encourages greater modularity and code reusability they take to it like ducks to water.

Learning OOP requires a commitment to training, and for best results, follow-up code and design reviews. It's also important to reward programmers for developing reusable code and for reusing it. I've found that with a one week training course, most programmers are comfortable applying object-oriented programming within about four to six weeks and are achieving higher productivity within two or three months. Reports of two to four times productivity improvements are about average, with some reports of ten-fold improvements, depending on the class libraries available.

MYTH: *Object-oriented programming is old wine in new bottles.*

In some cases, professionals react to news about object-oriented programming with skepticism. They've heard the benefits of lower software development costs for many other technologies and are hesitant to get involved in what might be a passing fad. With all of the new terminology and new languages it's easy to ignore OOP, passing it off as old wine in new bottles.

There is a lot of hype surrounding OOP these days, but it exists for a very good reason: object-oriented programming works. While object-oriented programming is no panacea; it can help manage the growing complexity and increasing costs of software development. Object-oriented programming is still in its adolescence, and it has the potential to improve productivity in the same way that structured programming did in the 1970s.

What is Object-Oriented Programming?

By putting the myths aside, we are able to define what is meant by object-oriented programming. For a language to be considered object-oriented it should meet the three criteria listed below.

- It must combine data and functionality together into a single logical unit, usually called an object.
- Objects must be able to inherit behavior from existing objects or their definition (An object's definition is often called its class.
- Different objects should be able to respond to the same message differently. This is also known as polymorphism meaning "many behaviors".

If we examine the current crop of object-oriented languages, including Actor, C++, CLOS, Eiffel, Objective C, Object Pascal, and Smalltalk, they all meet these three criteria. Languages which are sometimes called object-oriented, such as Ada or Modula-2, do not provide at least one of the criteria described above and should not be considered object-oriented.

These criteria help define object-oriented programming, but don't address issues such as where and how it can be applied. These are topics I will address in future articles.

Making OOP Payoff

There are a lot of myths concerning object-oriented programming, but luckily, increased programmer productivity is not one of the myths. By considering the use of OOP today, you would be considering an investment in training as well as the development of reusable software components. The return on this investment is lower software development costs and lower maintenance costs your organization will have.

Object-oriented programming is applicable to a wide variety of problems ranging from real-time control systems to graphically-oriented applications. Languages and tools are available to help you integrate OOP into your organization today, so that you can reap the benefits of greater software reusability in the future.

Zack Urlocker, *Manager of developer relations at The Whitewater Group, the creators of Actor, an object-oriented language for Microsoft Windows. Mr. Urlocker has written articles for several computer magazines and journals.*

Object-Oriented Analysis

by Peter Coad and Ed Yourdon *(Prentice Hall)*

As object-oriented programming moves from the research lab into main stream program development, software engineers as usual find a need to understand what their system must do, and how it will do it, before starting the coding phase. These questions are the domain of object-oriented analysis and design, the first of which is the topic of this book.

Some skeptics have voiced concerns that object-oriented development is only suitable for small to medium sized projects, since the what and how are more difficult in large systems. The first thing I noticed as I read this book was the authors had worked on VERY large systems (e.g., air traffic control) and understood the need to structure the development process but still take advantage of the power of object-oriented technologies.

In my experience, the advice in this book is sound. It's based on the wisdom that comes from experience. This experience tempers the authors enthusiasm, recognizing that the object-oriented approach, while valuable, has it's limitations. The authors point this out in the first chapter of the book, recommending that object-oriented analysis (OOA) be used along with other analysis tools: functional decomposition, data flow analysis, state-event-transition analysis, information modeling, etc.

The authors introduce the principles of object-oriented technology in a novel manner, developing the concepts from an encyclopedia's definition of classification theory. This demonstrates that object-oriented thinking has existed for quite some time in other disciplines and is not likely to disappear as the next hot topic comes along.

From these definitions, the book presents five steps of object oriented analysis:

Identifying objects;
Identifying structure;
Identifying subjects;
Defining attributes; and
Defining services.

The first two steps are quite common. As one might expect, an object is an encapsulation and abstraction of something in the problem space. The structure step identifies the classification structure (i.e., the IS-A relationships) and the assembly structure (i.e., the IS-PART-OF relationships) of the objects.

The third step, identifying subjects, is a process of organizing the objects into conceptual packages that aid in the reading of the specification document. The division into subjects is used to limit the reader's focus of attention, removing details at a time that they are unimportant.

The fourth step involves identifying the knowledge contained within objects and the fifth step develops the services (methods or member functions, etc.) required of an object.

With the presentation of each step, the authors give detailed instructions on how to accomplish the step. For example, the authors present two strategies to define services.

The first strategy observes that there are three different kinds of services: occur, calculate, and monitor. Occur services include the adding, changing, deleting, and selecting found in many objects. Calculate services involve computation upon the request of other objects. Monitor services monitor ongoing activities such as one might find in embedded systems. One looks at each object along with it's attributes and considers each of these kinds of services.

The second strategy is to analyze an object's life history and it's state-event-response behavior. In doing so, additional services become apparent. Analysis of an object's life history is significant and little has been written about it. It demonstrates that the authors have built real systems, not just research prototypes.

Unlike many technical books, this book is well written and easy to read; the first pass could be done in a few nights. But one is likely to re-read the book many times to grasp many of the fine details spread throughout.

I highly recommend this book, even though it does not answer every question that I might have on OOA. As an example, while directions to accomplish each step are most clear, in practice the steps are not addressed in a sequential manner; there is interaction between the steps, some jumping around, and iteration. I would like some guidelines on how one might make progress as the steps are tackled.

Other questions occurred to me, but I the authors addressed my concerns in their introduction. "OOA is a relatively young method; it will continue to evolve in practice...use this book as a starting point for applying OOA - tailoring and expanding the method to suit your specific organization or project needs."

Sounds like good advice to me. This book does not have all the answers but is a wealth of information that provides a good starting point.

Reviewed by
Norman L. Kerth
Elite Systems

Plug into the *Hotline* for trends, issues news and developments in object-oriented technology.

Subscribe today. Call 212-972-7055

A View from Britain

Interest in object-oriented programming and systems is growing worldwide. I would like to give a view from the other side of the Atlantic, and outline some of the research and teaching activity in the United Kingdom. This is a personal review, reflecting activities in which I am particularly interested.

Looking first at academic research work, there is much activity within research institutions, universities and polytechnics. Three categories can be identified.

First, there is work on the design of object-oriented languages and systems, such as MUST (University of Manchester) and Lingo (Linn Smart Products). There is widespread research utilizing highly parallel hardware using the object metaphor. Investigations of implementation techniques (both hardware and software) is continuing at Queen Mary's College (London), Linn Smart Products, and the University of Manchester.

Second, a growing body of researchers are using interactive programming systems such as Smalltalk and CLOS to experiment with new applications. For example, Smalltalk-80 is being used to implement a 'theorem-prover's assistant' called MURAL at the University of Manchester. Also at the University of Manchester, Smalltalk-80 is being used for the design of user interfaces for doctors. Additionally, Smalltalk is being used for experiments with 'design re-use' tools at the University of Lancaster.

Finally, object-oriented design techniques and object-oriented languages are being used in the construction of larger-scale research systems. An example of this is the work on distributed multimedia retrieval systems at the University of Manchester using C++.

Several academic institutions in the UK offer short, intensive training courses on object-oriented techniques intended for individuals working in the industry. This follows from a government initiative encouraging commercially-oriented short courses in Information Technology. For example, the University of Manchester offers training courses in object-oriented-design, Smalltalk-80 and Smalltalk/V, under its 'Post-Experience Vocational Education' programme. Manchester Polytechnic also offers Smalltalk courses. The University of Glasgow offers object-oriented DBMS, C++, and Smalltalk-80 courses. Companies also offer training courses: Praxis Systems in Bath offer object-oriented programming concepts and Objective-C courses, as an example.

Object-oriented programming and design techniques are also being taught at both undergraduate and postgraduate level. Many universities now include object-oriented languages somewhere within their syllabus. Increasingly, such languages are being used as primary teaching languages in undergraduate courses. As examples, the University of Southampton is teaching Smalltalk as their second-year language and the University of Salford is taking a similar teaching approach. Object-oriented systems are also widely used for undergraduate and postgraduate projects.

There is a large and growing community of users of object-oriented systems in the UK with every sign that this growth will continue. Add to this an active research programme and it is no surprise that there is now much awareness of object-oriented techniques in computer companies and research institutions, both large and small.

Dr. Trevor P. Hopkins
Computer Science Department
University of Manchester

Calendar

The HOOT calendar presents conferences and meetings that are focused exclusively on object-oriented technology. It should be noted that many other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to the Editor, HOOT, 8825 SW. Umatilla St., Tualatin, OR 97062

OCTOBER 17-18, 1989

The Object-Oriented Symposium—Presented by Digital Consulting, Inc. and chaired by Larry L. Constantine in Los Angeles. Contact: DCI, 6 Windsor St., Andover, MA 01810, (508) 470–3880

NOVEMBER 6-8, 1989

C++ at Work '89—Co-sponsored by The C++ Report and The Wang Institute of Boston University, Tyngsboro, MA Contact: Lisa Gennell, Wang Institute of Boston University, 72 Tyng Road, Tyngsboro, MA 01879, (508) 649–9731.

NOVEMBER 13-15, 1989

TOOLS '89—First International Conference in Paris, France. Contact: Jean Bezivin, Laboratoire d'Infomatique, Faculte des Sciences et Techniques, Universite de Nantes, 2 Rue de la Houssiniere, 44072 Nantes, Cedex 03 France.

DECEMBER 3-6, 1989

First International Conference on Object-Oriented and Deductive Databases—Kyoto, Japan. Sponsored by the Information Processing Society of Japan and the Advanced Software Technology/Mechatronics Research Institute of Kyoto. Features tutorials, technical sessions and exhibits. Contact: Won Kim at MCC (512) 338–3439

DECEMBER 4-8, 1989

International Workshop on Software Engineering and Its Applications—Toulouse, France. Contact: Jean-Claude Rault, EC2 269-287 Rue De la Garenne, 92000 Nanterre, France. Tel. 011–331–478070–00

DECEMBER 5-7, 1989

The Object-Oriented Symposium—Presented by Digital Consulting, Inc. and chaired by Larry L. Constantine, in Chicago. Repeat of the symposium presented in October 1989.

JANUARY 22-24, 1990

SCOOP-West—Techmart Center, Santa Clara, California. Co-sponsored by JOOP and the Wang Institute of Boston University. Contact: Bob Daniels, Wang Institute of Boston University, 72 Tyng Road, Tyngsboro, MA 01879, (508) 649-9731.

An Introduction to Object-Oriented Database Management Systems

by Tom Atwood

Object-oriented programming techniques are rapidly gaining support for a variety of applications where developers have discovered the limitations of conventional procedural programming paradigms. The benefits are straight forward; object-oriented programming reduces the cost and time required to produce application software by supporting the development of reusable code that is easier to maintain and enhance.

Like all good ideas, the concepts behind object-oriented programming are simple yet powerful. Object-oriented techniques raise the level of abstraction in programming through the use of two concepts common to all true object-oriented languages: encapsulation and inheritance (Figure 1).

But object-oriented programming by itself is not enough. Using an object-oriented database management system to build applications and store reusable code can save twenty to thirty percent of development cost beyond that achieved by object-oriented programming alone.

Historical Rationale for Database Management Systems

Prior to the availability of database management systems (DBMS), data was stored in files and it was the application programmer's responsibility to manage that data. Applications developed by different programmers managed the same data differently and it was difficult to ensure consistency for users and programmers. Database management systems simplify the job of software development by taking responsibility for:

- arbitrating the sharing of data among users;
- ensuring data integrity and recovery in the event of problems;
- distributing data in a network;
- managing the query process for large amounts of data.

The two essential features of object-oriented programming:

1. **Encapsulation,** the ability to store, with an object, all its attributes (state) and the operations it is capable of performing.
2. **Class Inheritance and Hierarchy,** the ability to pass on the attributes possible operation of a superclass of objects to a subclass.

Figure 1.

New Applications Need Object-Oriented Database Management Systems

An object-oriented database management systems provides the traditional services of a DBMS as described above, but treats data as objects. The first three generations of database management systems, hierarchical, network and relational, were designed to support alphanumeric data formatted into records. While this was appropriate and valuable for the typical MIS application, it left a void for mechanical computer-aided design (MCAD), electrical computer-aided design (ECAD), computer-aided publishing (CAP), computer-aided software engineering (CASE), and advanced office automation applications. These applications rely on complex, non record-oriented data such as graphical data, two and three dimensional information, images, voice and video data.

The first generation of these data-rich applications implemented proprietary file systems to support required data types and to automate the work of the individual design engineer, the individual author, or the individual programmer. The second generation systems now focus on automating a design team—a VLSI design team, an editorial team or a programming team and has generated the need for a database management system that can support:

- version and configuration management for designs as they evolve;
- concurrency control for cooperative work among a team of professionals;
- backward compatibility so designs done using the previous release and those done with the new release can coexist in the same database, and the old designs can be processed by the new software.

At the same time, the object-oriented DBMS often require performance levels two orders of magnitude faster than existing DBMS technology. It is this tension between the need for a full suite of traditional DBMS capabilities and very high performance that makes the question of implementation architecture a key one for object-oriented DBMS design. The basic question is — How can you possibly get it to go that fast?

The second generation of object-oriented database management systems under development now, Object Design's ObjectStore among them, have taken a radically new tack in their implementation architecture. Internally, they look much more like the map-it-into-virtual memory architectures underneath the proprietary file systems used in current computer-aided design applications.

Specifically, these next generation architectures are using sophisticated memory management techniques to maximize the data management operations that occur in memory. They are being built using the latest client-server architectures and perform most calculations on the client workstation to gain the maximum use of available compute cycles. These architectures minimize network traffic and disk access through the use of caching and clustering.

Finally, by storing objects in a DBMS, the operations that objects of a given type are capable of performing are known to the DBMS and the DBMS can exploit this knowledge to optimize performance. Performance tools can be built to monitor reference trace patterns for frequently used operations, and do anticipatory 'prefetching' of objects the program is about to reference.

Benefits of Object-Oriented Technologies

Sharing, distribution, and high performance will be the initial reasons object-oriented DBMSs are used, but higher level abstraction will be the reason object-oriented technologies dominate the 1990s. Object-oriented programming makes it easier to model real world entities, such as an airplane wing, or concepts, such as a software module.

Now, for the first time, the developer can describe entities in terms closer to their real world analogs and keep the descriptions formally distinct from the computer representation. The application or database designer models his world in terms of the same primitives he normally uses to think about it-objects, attributes of those objects, relationships between objects, and the operations which can be done on objects of a particular type. Several advantages fall out of implementing an object-oriented conceptual model. The key ones are:

(1) highly efficient type specific representations
(2) insulation of applications from changes in these representations through encapsulation.
(3) guaranteed consistency between the operations executed to objects of a given type by different applications
(4) dramatic reductions in the cost to develop new applications because of the ability to inherit code from object types already defined in the database

AT&T's Bell Labs experience with object-oriented technologies is illustrative of the potential benefits. During a three year period, Bell Labs moved new development from C to C++. Their analysis of C programs showed that typically 80% of the code for a given application was new code, written explicitly for that application; only 20% of the code came from preexisting libraries. In stark contrast, analysis of C++ programs showed that by the third year of C++ use, only 20% of the code was new code written explicitly for the application served by the new program; 80% of the code was inherited from object types already defined in the type library. That represents a 4:1 reduction in the lines of new code that had to be written to implement a new application. After accounting for design and debugging time, total development time was often decreased by 60%.

Can Relational Databases Support Objects?

To date, most attempts to store complex data in relational databases have been implemented by putting a pointer into a record that points to the data stored as a BLOB (Binary Large Object). But all the detailed structural information describing the individual small component objects is lost. Design applications would still have to manage their own data in files and these files would be stored in the relational DBMS. While this is viable for some business requirements it fails to serve the needs of high performance interactive graphic design applications.

Others have attempted to decompose the complex graphical data into record formats that can be stored in relational DBMSs. While technically feasible, the performance overhead associated with accessing this data in a manner consistent with interactive design applications make these implementations unworkable.

Initially, object-oriented DBMS technology will initially not compete with record-oriented DBMS. Rather, object-oriented DBMS products will meet the needs of applications not served by record-oriented DBMS technology and expand the entire DBMS market.

The Growing Need For Standards

An ideal object-oriented DBMS should fully support object-oriented design and programming. It should integrate into commercial computing environments, providing platform independence and interfaces to existing applications and databases.

The next generation of object-oriented DBMSs will be Unix-based, support the emerging user interfaces standards, and focus initially on support for development in C++, the defacto standard for object-oriented programming.

In the future, a data model standard will emerge that can be mapped into the major commercial languages and object-oriented DBMSs. C, Pascal, Lisp and ADA already have object-oriented derivatives. Microsoft has announced an object-oriented Basic and research on an object-oriented COBOL is underway at Hewlett-Packard for presentation to the Codasyl group which feeds the ANSI standards efforts for COBOL.

Is this enough? The when, where, how and why of standards is an important issue whose resolution can help resolve accelerate the acceptance of object-oriented database management systems. In the next column we will discuss the need for standards in more detail.

Tom Atwood., *Founder and chairman of Object Design, has pioneered the commercial development of object-oriented technology. Previously, he was one of the founders of Ontologic, one of the first companies to introduce a commercial object-oriented DBMS.*

EurOOP: Object-Oriented Activities in Europe

OOP is big news in North America; it is fast becoming the only news in Europe. This column will cover some of the issues, events, and people involving OOPs in Europe.

ECOOP—Whose needs are being served?

To open up some topics that seem to be making waves right now, we look at the recent European Conference on Object Oriented Programming (ECOOP), held in Nottingham University, England, from July 10th to 14th, 1989.

The biggest question is familiar from other fields of computing; how far are the purely academic objectives of many leading OOP researchers in tune with the needs of workers in real applications?

ECOOP was a definite success in terms of numbers attending, with around 400 participants. It offered a varied but scientific program, which was the primary objective of its organizers. It attracted large numbers from both academic and industrial backgrounds. To some this supports the notion that a purely scientific conference is what everyone wants.

On the other hand, it was quite clear that many had come to learn, not to debate matters of theory. For the first time tutorials and workshops were a part of the conference with the latter running in parallel with the technical sessions. Like the experiences at earlier OOPSLAs, these very practical oriented sessions were a major draw. As an organizer of a workshop on windowing systems and OOP, I was apprehensive that few would come, since they would be missing the main speakers, and worried that I would be missing the real action. Yet in the two workshops I attended (the other was on object-oriented approaches to simulation) the room was packed with around a hundred people and the discussion was lively.

The message from ECOOP for me was that although it is important to discuss strong typing, formal semantics, and the other issues dominating the academic world, the needs of those trying to get the benefits of OOP techniques are simpler and addressing their needs is an urgent matter.

Bringing OOP to the masses?

Addressing these needs is underlined by the plans announced by various groups at ECOOP. Here I merely list those which came my way.

TOOLS 89 is aiming to provide a combination of tutorials and more formal papers focussing on applications of OOP techniques. Although backed by Bertrand Meyer's Interactive Software Engineering Inc., it boasts an independent program committee and will cover a range of languages. Details can be obtained from Jean Bezivin, Laboratorie D'Informatique, Faculte des Sciences et Techiques, Universite de Nantes, 2 Rue de la Hoursiniere, 44072 Nantes Cedex 03, France.

Also definite is OOPS 30. This is being run by the OOPS Specialist Group of British Computer Society and says that it aims to bring together practioners and discuss applications and real experience. Details from Bruce Anderson, ESE, University of Essex, Colchester, Essex, England

Gazing into the limbo of 1990, when ECOOP will be visiting OOPSLA in Canada, two further practioner events are being lined up.

SCOOP is visiting London in the early summer, following its success in North America. HOOT will be keeping you in touch with plans for this.

An event called STOOP (how many acronyms are there to go?) is planned for September in Berlin. This aims at bringing the big names to the real users. Informal agreements from Meyer, Nygaard, and Stroustrup appear forthcoming. Details from Ron Kerr, Computing Laboratory. The University, Newcastle upon Tyne, England.

In the early hours of a morning in a smoke filled back room at ECOOP, I was asked if the demand for OOP related events could support the number being planned. My answer then as now was "Without a doubt."

Who uses OOP in Europe?

The move to new technologies is always uneven and dependent on a wide range of factors. The large number of people from large companies attending OOP events is an apparent indication that OOP is being seriously used, but how far is the current curiosity turning into real, long term commitment? Certainly parts of major corporations such as British Telecom and Thomson seem to be going the whole way, but who else out there is really involved?

EurOOP would like to hear about the experiences of users, both good and bad. How useful is OOP really turning out to be? How realistic are the claims? What is missing to encourage its use?

If you have something to say about object-oriented programming regarding these issues in a European context or just some news about the field, let me know.

Rob Pooley, *Lecturer at Edinburgh University and the Managing Director of Edinburgh Application Software Engineering Ltd, a company specialising in implementation and use of OOP products. Contact Rob Pooley, EurOOP, 5 Marine Parade, North Berwick, Scotland.*

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes,*** I want to be plugged into the latest thinking, and developments in the object-oriented technology. Immediately enter me as a charter subscriber at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	2 years (24 issues)
☐ ~~$299~~ $249	☐ ~~$598~~ $498

outside the U.S.—add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT—U.S.currency drawn on U.S. bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card #_____ Exp. date_____
Signature _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____

For immediate service—call the Hotline 212-972-7055 or Fax 212-370-4983

HOTLINE on
Object-Oriented Technology
The International Newsletter

310 Madison Avenue
Suite 503
New York, NY 10017

BULK RATE
U.S. POSTAGE
PAID
MENASHA, WI
PERMIT NO. 47

0001173 ACM
K RUBIN
PARK PLACE SYSTEMS
1550 PLYMOUTH STREET
MOUNTAINVIEW LA 94043

HOTLINE on
Object-Oriented Technology
The International Newsletter

December 1989 **Volume 1, Number 2**

OOPSLA '89

by Jeff McKenna

New Orleans. October 2 – 6 1989. They came in bunches. Over 1,950 of them. All headed for the Hyatt Regency, the site of ACM's fourth annual conference on Objected-Oriented Programming, Systems, Languages, and Applications.

Without doubt the world's largest conference on object-oriented technology, the growth of OOPSLA over the past four years is indicative of the growth of interest and activity in the field. As in years past, OOPSLA presented the opportunity for beginners and experts alike to get together, learn the latest developments, discuss and argue the issues of the day, and meet the movers and shakers in the field.

The Numbers

First, let us examine those who attended OOPSLA this year. We were able to get a peek at the results of the survey of attendees at this year's conference, and from this data we can piece together a typical profile. Note that this survey was not conducted in a scientific manner, only about 5% of the attendees were polled. It does reasonably appear to reflect what we observed.

OOPSLA is primarily a conference for researchers and developers in object-oriented technology, and this was reflected in the numbers. Some 45% of the attendees classified themselves as researchers and 26% as developers. Programmers were strongly represented with 28% being systems programmers and 33% applications programmers. More than 13% of the attendees were in management, about 8% of the attendees were students, and 4% were marketers. (These total over 100% since many were in more than one category.)

It is clear from the numbers that the attendees are actively engaged in object-oriented technology. More than 80% of them stated that they are currently using the technology. OOPSLA is also a place to learn, and 16% of the attendees were considering using the technology. The remaining 4% were there to learn but were not considering its use.

OOPSLA attendance appears to be addictive; 41% of this year's attendees attended last year's conference, 38% attended in 1987, and 25% were at the first conference in 1986. The total attendance of the 1986 conference was 1,000, 1,200 in 1987, 1,500 in 1988, and 1,950 this year. Very strong

continued on page 3

Contents

Feature:
OOPSLA '89
1

Thoughts from the Editor
2

Note from the Publisher
2

Guest Article:
C'ing into the Future
5

Publicity Corner:
O-O Technology in the News
7

Calendar
7

Data Bases:
Standards for OO-DBMS: A Manager's Guide
8

Passing Messages
11

Analysis and Design:
Synthesis: An Object-Oriented Analysis and Design Method
12

Company Showcase
14

Book Review:
Object-Oriented Environment in C++: A User-Friendly Interface
15

Languages and Environments:
From Languages to Environments
16

Product News
18

Thoughts from the Editor

Welcome to the second issue of the Hotline on Object-Oriented Technology. We were very pleased with the response to HOOT at the introduction at OOPSLA. One attendee commented that if she had seen HOOT she would have not have needed to attend the conference! While this seems a bit much, we understand the thought. In this issue we continue to give you the up-to-date information in the object-oriented field.

This month we feature an article reviewing the recent OOPSLA conference in New Orleans. This conference brings together all the major players in object-oriented technology and we felt that a complete report would be of considerable interest to you. As an addendum, we have included a commentary by Keith Howard on the 'language wars' that have been a big part of OOPLSAs. His comments complement a most interesting article by Ed Yourdon in the latest issue of American Programmer. We have asked Ed to comment on the 'objectifying' of COBOL.

In our regular feature on *OO Analysis and Design*, we are presenting Synthesis, a design approach that contrasts with last month's article on analysis and design. Zack Urlocker continues the *OO Languages and Environments* series by introducing the issues of environments.Tom Atwood continues the *OO Data Base* column with a discussion of the role of standards in OODBs.

A new column that we have added is *Passing Messages*. In this column we will give you the small bits of gossip and information that we overhear in our wanderings. If you have any tidbits that would be appropriate, drop us a line.

Our *Company Showcase* continues in this issue and the *Product News* section is very full. Commercial activity continues to grow at a remarkable pace.

We have two particularly interesting happenings in the pipe for next month.

Tom Love will be giving us a report of the GREED programming competition that was held in OOPSLA. Nearly a dozen implementers participated in the competition and Tom's article should be good reading.

Adele Goldberg of ParcPlace Systems will begin a six-part series on the management of object-oriented development that promises to be very interested. Her many years of managing object-oriented projects makes her uniquely qualified to address the topic.

Now that you have had a chance to read a couple of issues of HOOT, let us know how we are doing. There is a great deal of controversy in the entire field and we mean to show as many sides as possible.

Jeff McKenna
Editor

Note from the Publisher

Jecrit cette lettre from a sidewalk café in Paris. I'm here attending TOOLS '89, an important international object-oriented conference. Holding this conference at the recently re-opened CNIT, the striking hi-tech exposition center in the corporate section of Paris, is a big step towards commercializing and legitimizing O-O technology in Europe. Over 400 are here to learn more about OOT. The organizers are to applauded for thinking big.

Our sister publication, *The C++ Report*, recently co-sponsored the first C++ AT WORK '89 conference at the Wang Institute of Boston University. Over 450 developers gathered to anxiously hear C++ gurus Bjarne Stroustrup, Andrew Koenig, Zortech's Walter Bright and Glockenspiel's John Carolan spout words of wisdom about developments in the language. Andrew Koenig's tutorial attracted a capacity 240 attendees.

C++ AT WORK '89 was designed to promote and stimulate C++ in business. The audience was split equally between active users and those wanting to learn. Several large software manufacturers attended attempting to gauge interest and committment to C++. The consensus, (though a biased sample), was that C++ is destined to become a mainstream language by 1992 as the number of current users, about 50,000, doubles every year and the 300,000 active C programmers migrate up. Major software companies are gearing up for an aggressive launch of C++ products in 1990.

The conference proceedings can be purchased for $39 by calling 212-972-7055.

Richard P. Friedman
Group Publisher

HOTLINE on
Object-Oriented Technology
The International Newsletter

Jeff McKenna
EDITOR

ADVISORY BOARD

Jim Anderson
Digitalk, Inc.
Tom Atwood
Object Design, Inc.
Grady Booch
Rational
Larry Constantine
Consultant
Cort DeVoe
IBM
Adele Goldberg
ParcPlace Systems
Bertrand Meyer
Interactive Software Engineering
Reed Phillips
Knowledge Systems Corp.
Trygve Reenskaug
Taskon A/S
Bjarne Stroustrup
AT&T Labs
Dave Thomas
Object Technology International
Zack Urlocker
The Whitewater Group
Eugene Wang
Borland International

Richard P. Friedman
Group Publisher
Christine Coleman
Art Director
Philip Desiere
Technical Advisor
Melanie Younossi
Marketing Assistant

The Hotline on Object-Oriented Technology (ISSN#1044-4319) is published monthly by SIGS Publications Inc., 310 Madison Ave., Suite 503, New York, NY 10017. (212) 972-7055.

©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher.

Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30. Send address changes and subscription orders to HOOT, 310 Madison Avenue, Suite 503, NY, NY 10017
Submit editorial correspondance to Editor, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062, or call (503) 692-1889.

operators such as "+" and "+=" have been extended to include vectors and matrices. Standard C functions such as "cos()" and "sin()" have also been extended. Many new functions for statistics and numerical modeling are provided as well. A complete complex number class is provided. Fast Fourier Transform server classes allow users to take the FFT or inverse FFT for any length series (real or complex) quickly and easily. Using the inheritance property of C++, new classes to do specialized tasks can easily be created from the Rogue Wave classes.

A complete, 120-page User's Guide and Reference Manual is provided and full source code is included. The classes compile under a variety of C++ compilers (V1) under both MS-DOS and Unix.

The classes cost $150 and are available immediately.

An enhanced version with graphics capability is planned.

For more information, contact:
Dr. Thomas Keffer
Rogue Wave
P.O. Box 85341
Seattle, WA 98145-1341
(206) 523-5831

Object-Oriented Library for Turbo Pascal 5.5

TurboPower Software announces Object Professional 1.0, an object-oriented library that enables Turbo Pascal users to write powerful applications more quickly. Object Professional contains over 100 high-level object types with over 1000 methods, including user interface object types, data object types, and system oriented routines.

The user interface object types provide multiple overlapping and resizable windows. The objects are a logical, extensible hierarchy, which provide generality and power with minimal overhead and maximum code sharing. Windows support scroll bars and mouse selection, flexible borders, color selection, and multiple title strings. Other supporting objects provide menus, text editing, scrolling data entry screens, pick lists, and help capability.

The data object types provide proven and documented objects such as stacks, linked lists, circular lists, string dictionaries, and virtual arrays.

The system oriented routines provide support for TSRs, including swapping TSRs, interrupt management, EMS management, DBIOS and DOS capabilities, and a swapping Exec manager.

Object Professional 1.0 costs $150, including complete source code, three volumes of documentation, and a royalty-free license for commercial development. It will be available in December 1989.

For more information, contact:
Robert Benson
Director of Marketing
TurboPower Software
P.O. Box 66747
Scotts Valley, CA 95006-0747
(408) 438-8608
To order call (800) 333-4160

Object Plus

EASYSPEC, Inc., announces Object Plus, a practical, easy-to-use, object-oriented designer and C++, C, and Ada code generator.

Object Plus is a MS Windows based CASE tool, based on object-oriented methodology which: 1) shortens the time and cost of development by generating reusable software components; 2) lowers the cost of maintenance by utilizing the concepts of modularity and information hiding; and 3) supports the full software lifecycle, i) requirements definition, ii) preliminary, system, and detailed design, iii) code generation, and iv) documentation.

The major functions in Object Plus are: 1) identification of the application's major functions; 2) specification and documentation of the application requirements; 3) identification of specification and design of the application objects; 4) establishment of an element dictionary and object repository; 5) code generation for C++, C, and Ada; 6) data modeling supporting major DBMS systems; 7) reverse engineering to import existing code; and 8) provision various reports.

Object Plus runs on an IBM AT or 100% compatible using DOS 3.0 or later, EGA/VGA or better, and MS-Windows 2.0 or later. It is priced from $995 to $6,700, depending on the number of users, and it is available now.

For more information, contact:
Anthony A. Lekkos
EASYSPEC, Inc.
17629 El Camino Real, Suite 202
Houston, TX 77058
(713) 480-3233
Fax (713) 480-6606

Objectworks for Smalltalk-80 on HP 9000 Series 300 Workstations

ParcPlace announces the signing of a marketing and development agreement with Hewlett-Packard's Software Engineering Systems Division. This agreement allows ParcPlace to develop and support additional features on the Objectworks for Smalltalk-80™ programming system specifically for the HP 9000 Series 300 workstations. A major feature will be the running of Objectworks under X-Windows.

ParcPlace indicates that HP currently uses Objectworks for Smalltalk-80 as a training tool and has two internal applications using the technology: OSA - an object-oriented systems analysis tool and HPM - the Hierarchical Process Management tool for manufacturing simulation.

Objectworks for Smalltalk-80 on HP 9000 Series 300 will be available in the first quarter of 1990 and is priced at $3495. ParcPlace currently supports the HP Vectra.

For more information, contact:
Helene Tannor
ParcPlace Systems
1550 Plymouth Street
Mountain View, CA 94043
(415) 691-6700
Fax (414) 691-6715

Gemstone has C++ and Relational Database Links

Servio Logic Development Corporation announced two new interfaces to Gemstone, its object-oriented database product. The interfaces are to C++ and Sybase's relational database via SQL.

The C++ interface provides persistent storage capabilities to C++ programmers. The interface is implemented as a preprocessor and based on the standard C++ syntax (V2.0). In addition to the interface, the product provides a proven class library modeled on the Smalltalk class library developed at Xerox PARC.

The C++ interface will be available in the first quarter of 1990 and will be available to Gemstone customers at no charge.

The SQL interface allows users to use data stored in both object-oriented and relational databases within the same application. The Gemstone/SQL link provides for queries of the relational database from within Gemstone and the Sybase response to be returned to Gemstone. Servio plans to provide interfaces to other relational databases in the future.

Beta versions of the SQL interface will be distributed later this year with product shipments beginning in the first quarter of 1990. The price of the interface is $1000.

For more information, contact:
Mark Boyd
Servio Logic Corporation
1420 Harbor Bay Parkway, Suite 100
Alameda, CA 94501
(415) 748-6200

Plug into the *Hotline* for trends, issues, news and developments in object-oriented technology.

Subscribe today. Call 212-972-7055

Product Announcements should be sent to the Editor, HOOT, 8825 SW Umatilla St., Tualatin, OR 97062.

Zortech C++ V2.0

Zortech Inc., of Arlington, MA, announced the release of its C++ V2.0 Developer's Edition for MS-DOS. This new product is compatible with the AT&T V2.0 specification, including new features such as multiple-inheritance and type safe linkage.

The Zortech C++ compiler transparently supports usage of EMS (LIM) in developed applications. A seamless edit/compile/debug environment is included, which incorporates a new SAA/CUA type user interface. The system feature a new C++ Debugger, full standard library source code, MS-Windows programming compatibility, object-oriented graphics classes, improved resident context sensitive help, and a sophisticated TSR library which allows most applications to be made resident with a single function call. The package features extensively expanded and improved documentation in 5 volumes.

The compiler is priced at $199.95 and the debugger, library source, and tools are priced at $149.95 each. These four are packaged together together with an environment in the Developers Edition, which is priced at $450. Multiuser packs are available at reduced rates.

For more information, contact:
John Haggins
Zortech Limited
106-108 Powis Street
London SE18 6LU
England
(44)-1-316-7777
Fax (44) 1-316-4138
or:
Paul Leathers
Zortech Incorporated
1165 Massachusetts Avenue
Arlington, MA 02174
(617) 646-6703
Fax (617) 643-7969

ONTOS

Ontologic has announced ONTOS, the first multiuser object-oriented database based on C++. ONTOS is one member of a family of object-oriented database products:

ONTOS- a multiuser, distributed, full-featured object database;

ONTOS Persistent C++ — a standalone single-user version; and

ONTOS Class Library — a set of extendable classes, including aggregates such as sets, lists, dictionaries, and arrays.

ONTOS's modular design lets developers configure an object database in accordance with their specific requirements. Together or separately, these modules combine the high productivity of object-programming with the persistence, sharing, integrity, and query ability of a database. ONTOS incorporates advanced database sharing and integrity features, including long transactions, nested and shared transactions, versions, and both optimistic and pessimistic concurrency control.

Ontologic also announced that Index Technology will be using ONTOS in a future release of their product Excellerator.

ONTOS is currently available for SUN and Apollo workstations and supports C++ and SQL. Development is underway for the OS/2, DEC VAX, and DECStation platforms. ONTOS is priced at $15,000 and is available now. ONTOS Persistent C++ will ship in December and its price is $9,900. ONTOS Class Library will also ship in December and is priced at $695.

For more information, contact:
Robert Martin
Ontologic Inc.
Three Burlington Woods
Burlington, MA 01803
(617) 272-7110
Fax (617) 272-8101

NEXPERT OBJECT™ and ONTOS

Neutron Data and Ontologic have announced a bridge between NEXPERT *OBJECT*™, Neutron Data's expert system shell, and ONTOS, Ontologic object-oriented database. With the bridge, application developers can map ONTOS's object-oriented data structures into the NEXPERT *OBJECT* representation, which can then be intelligently reasoned upon with NEXPERT rules.

It is anticipated that the bridge will be used in application areas such as computer-aided design, computer-aided software engineering, network management, communications, and office automation. Users of the bridge may create objects in NEXPERT *OBJECT* and have the corresponding objects automatically created in ONTOS as well as having ONTOS objects automatically create corresponding NEXPERT *OBJECT* objects. The bridge combines the power of object-oriented databases with the reasoning power of a rule based system.

The NEXPERT and ONTOS bridge will be available first on the SUN and Apollo platforms, followed by OS/2 and DEC platforms. No pricing or availability information is yet announced.

For more information, contact:
Robert Martin
Ontologic Inc.
Three Burlington Woods
Burlington, MA 01803
(617) 272-7110
Fax (617) 272-8101
or:
Rico Bumbaca
Neuron Data
444 High Street
Palo Alto, CA 94301
(415) 321-4488

OOSD/Ada Design Editor

Interactive Development Environments, Inc. (IDE) announced the addition of the OOSD/Ada Design Editor to the Software through Pictures® family, providing Ada language support for the standard Object-Oriented Structured Design (OOSD) notation. This is the first CASE product to provide explicit support for design reusability — the Ada Reuse Library assures quality and productivity by easing reuse of proven design elements. Also included is the Ada Guidance System, which minimizes errors during the creation of a model.

OOSD/Ada Design Editor is built upon the base components of IDE's Software through Pictures multiuser CASE environment, which includes a data dictionary, a document preparation system, a multiuser locking facility, version control, and links to other products. The guidance system minimizes errors by enforcing Ada specific syntax rules between interacting components.

The OOSD/Ada Design Editor will be available in the first half of 1990 on a range of workstations including SUN, Apollo, DEC, and HP. No prices were announced.

For more information, contact:
Nobby Akiha
Director of Marketing Communications
Interactive Development Environments
595 Market Street, 12th Floor
San Francisco, CA 94105
(415) 543-0900
Fax (415) 543-3716

OODBMS Report

The EBDE Project announced a report on object-oriented databases. This report evaluates seven object-oriented database management systems for their suitability in the engineering domain.

The databases covered are: Vbase (Ontologic), O2 (GIP-Altair), Generis (Deductive Systems, Ltd.) POSTGRES (University of California), SDS2 (Gerrard Software, Ltd.), TORNADO (Metis A/S), and PS-Algol (University of St. Andrews/University of Glasgow). Three test cases are used in the study: Bill of Parts and Constraints, Geometry, and Process Modeling.

The report is 222 pages and costs $153.

For more information, contact:
Paul Barbour
EDBE Project
P.O. Box 84
Plymouth PL1 1SZ
England
0752-673814

Numeric Class Library for C++

Rogue Wave announces a set of object-oriented numerical tools that allows developers to perform complicated operations on vectors and matrices as easily as with built-in types. All of the standard C arithmetic

continued from page 1

growth.

Many people are interested in discovering which object-oriented language to use and the OOPSLA attendees voted strongly for C++ and Smalltalk, with 51% indicating that they are using C++ and 46% using Smalltalk, and a fair number indicating that they are using both. All the remaining languages garnered less than 10% each, Object-Pascal led these with 9%. We believe that the Pascal camp will grow strongly since implementations are just beginning to be available.

The last set of numbers represent the exhibits. These were 46 companies represented occupying 80 booth spaces in the exhibit hall. This is a huge increase over the first conference in 1986 at which there were 13 companies with 22 spaces.

The Activities

As with any large conference, there were a number of activities to keep the attendees busy. At times, OOPSLA appeared to be a three-ring circus with many more opportunities than time.

The conference started on Sunday (along with a Saints football game!) with the first of five workshops on topical activities. These workshops were by invitation only and provided an opportunity for a small number (fewer than 50) of practitioners to discuss specific topics. The topics of this year's workshops were: "Designing Software for the Reusable Component Market," "Objects Without Classes," "Domain Modeling for Software Engineering," and "Common Lisp Object System (CLOS)." Workshop attendees were pleased with these workshops. Look for ACM's SIGPLAN to publish the results of these workshops.

The tutorials were held on Monday and Tuesday. This year there were some 15 tutorials covering a wide variety of object-oriented technology topics and skill levels.

All tutorials except one were sold out with over 2,300 attendees (clearly, folks attended more than one tutorial). One observation of note is that the language introduction tutorials were less in demand this year. The hot topics were OO analysis and OO design, OO databases, and the engineering/management of OO software. The overall quality of the tutorials was good, but not excellent. We hope that at next year's conference, the quality will improve with special attention paid to the presentation skills and material preparation.

OOPSLA '89 had a full schedule of demonstrations of work in progress from researchers. While the technical content of many of these was noteworthy, attendance was limited. The physical layout of the hotel did not help in this regard.

Birds-of-a-feather (BOF) sessions (small spontaneous meetings of parties of like mind) have always been an important part of OPPSLA. The publicity of this year's BOFs was poor and the meeting rooms were small. There were also so many scheduled activities that it made scheduling BOF meetings difficult.

One meeting of note was the GREED BOF. This meeting had presentations by about a dozen implementers of a simple game called GREED in a variety of languages and systems. Each presenter summarized his design and implementation. The designs were remarkably similar. We will be having a full report of the GREED experience in next month's HOOT.

Another BOF of interest was between the vendor and users of Objective-C, C-talk/Views, and Complete C. Each of these languages is a C hybrid, each has a different syntax, and each has a small share of the language market. They first met earlier this year to discuss the joining together of their languages into a single syntax. These discussions continued at OOPSLA, and will continue in Europe at TOOLS '89. We applaud these efforts and encourage them to come to an agreement. Such a language could be a reasonable alternative to the extremes of C++ and Smalltalk. We suspect that NeXT and IBM might like this alternative also.

Activity surrounding OOPSLA is also of interest to those keeping an eye on the technology. There were 15 other meetings focused on OOT sponsored by companies. As examples, NCR had a day-long meeting and various user groups held meetings.

The social activities of OOPSLA were a big success. Reception were held every evening with the welcoming reception a gala affair, including an ice sculpture and live music. OOPSLA attendees drank all the beer in the Hyatt one evening! OOPSLA made it very easy to schmooz with one's friends. For those wanting to get out, New Orleans provided ample opportunity for partying.

One major disappointment in the social area was the banquet. The banquet itself was fine, but there was no banquet speaker. Apparently the speaker had cancelled at the last moment. The organizing committee did manage to piece together a short program, including some amusing anecdotes and a slide show of the conference, with some hokey music. Barely adequate.

The Program

The heart of the conference is the technical program and the keynote address kicks it off. Dr. Peter Wegner of Brown University discussed the Conceptual Evolution of Object-Oriented Programming. While the topic was of considerable interest, it was an uneven presentation and the visual materials were inadequate for the

nearly 2,000 people in the room.

The remainder of the technical program consisted of paper presentations and panels.

This year's papers covered the usual issues of OOPSLA papers: inheritance, reflection, delegation, agents, and concurrency, to mention a few. The general consensus was that this year's papers were of a slightly lower quality than in previous years. No single paper was outstanding, although the paper on the language SELF received more attention than the others.

All 45 papers are printed in the conference proceedings and the proceedings received the very highest marks from the attendees. For those interested, copies of the proceedings may be obtained from ACM Press.

Nine panels were held as part of the technical program. They ranged from star gazing (the future of OOP) to the technically detailed (the role of transactions in OO systems). One panel held at 7:00 am (!!) on prototyping was remarkably well attended.

The first and the last panels received the most attention. To start, Esther Dyson (Release 1.0) moderated a well-attended panel of Bill Joy (Sun), Bertrand Meyer (ISE), and Dan Weinreb(Object Design), discussing the future of OOP. The most interesting question of the conference was when Esther asked the audience if the world should standardize on one OO language. Not a single person raised his hand! There also was much poo-pooing of OO COBOL, which is not a bad idea for all those COBOL programmers out there.

The last panel was moderated by Daniel Bobrow (Xerox PARC) with Peter Deutsch (ParcPlace), Gregor Kiczales (Xerox PARC), and Bjarne Stroustrup (Bell Labs). Each of these three men has implemented a major OOlanguage, and was asked to reflect on the advantages of the other languages. The result? It wasn't clear, but the goals and objectives of each language is clearly different. The three gentlemen could see the advantages of the other languages more clearly than many of their more vocal adherents.

The Exhibits

The measure of the growth of OOP is nowhere more clearly shown than by the growth in number, size, and sophistication of the exhibits at OOPSLA.

As mentioned, this year there were some 46 companies in 80 booths. While it is difficult to cleanly classify some of these companies, one rough breakdown is this:

OOAnalysis and Design/CASE	3
OO Language and Environments	18
OO Data Bases	5
H/W Vendors	6
Training and Consulting	5
Booksellers and Magazines	9

The area of OO Analysis and Design is the area of the greatest potential growth. The presence of CADRE is the first indication of CASE vendors moving into this area. We did not see much at the CADRE booth that was object oriented but we believe that they will learn. Interactive Development Environments was showing their OOSD diagramming approach. Project Technology was discussing their OOA techniques, which are not currently supported by any software product. While not showing in a booth, Chen & Associates was discussing their Object-Designer, which aids in OO Analysis using graphical techniques based on ER modeling.

OO Languages and Environments was the biggest group with most of the known vendors and some new ones. Rather than merely listing all the vendors and their products (watch the Product News section of HOOT for new announcements!), we will note some of the new and more interesting products we saw.

The newly announced Smalltalk/V PM by Digitalk was considered by many to be the best new software at the show. We consider this product important enough to be featured in a product review next month.

Software Productivity Solutions, Inc. was showing Classic-Ada, which brings object-orientation to the world of Ada programming. This may play well with the growth of OOAnalysis and Design techniques targeted towards Ada.

ObjectVision, Inc. introduced ObjectVision, which allows users to develop OOapplications graphically with automatic code generation to C++ or Turbo Pascal 5.5. TGS Systems, Limited, demonstrated another graphic-based programming system, Prograph.

The presence of Neuron Data emphasized an AI-based company's commitment to OO technology. Their connection to Gemstone and ONTOS was highlighted. Nantuckett, Inc. was another new player illustrating the movement of OO technology into application-based products. Notable in their absence on the exhibit floor were Glockenspiel, Zortech, and Microsoft.

The OO database vendors presented a wider range of products than was presented last year. Servio-Logic remains strong with their product Gemstone, now offered with alternative interfaces. While Object Design

A fast interactive environment makes component testing more convenient and reduces the amount of testing required during system integration.

Design . . . and Experiment!

During the implementation stage of any project, there is always a need to experiment with different approaches. For example, you may wish to see which design or algorithm is most efficient for representative sets of data. Experimentation is no substitute for good design, but it can allow you to cope with changing requirements, unknown data, and other uncertain elements in a more effective way.

An object-oriented programming language, coupled with an interactive environment, can be a powerful means for dealing with complex, unstable problems. Ideally the environment should allow the programmer to modify the internal design portions as long as they do not affect module interfaces.

During development of medium and large object-oriented projects there is a need to reorganize classes in the hierarchy and to factor out common elements. This extra effort often makes the difference between a solution to the current problem, and a general reusable one. At the very least, the programming environment should make it easy to reorganize classes by changing the ancestor class their instance variables. When reorganizing the class hierarchy is difficult, productivity suffers and code reusability is compromised.

Inspecting and Debugging

Object-oriented programming is based, in part, on combining data and functionality together into objects. Because objects are more than just data, its important to be able to examine them and test them, just as we do with source code. An inspector is a tool which allows us to examine the contents of an object and its instance variables. Use of an inspector can eliminate the need to write debugging code to display objects.

Ideally, the inspector should display the objects in high-level terms, with which the programmer is familiar, rather than the internal storage format. For example, if an object contains a date instance variable, it should be displayed in a meaningful format, (e.g., 10/12/1989) rather than as a hexadecimal or binary number. Of course, the low-level format should be accessable to the implementor of the class. The inspector should also allow the programmer to modify objects at any time. This makes it easier to recover from errors during testing.

Debugging requirements in an object-oriented language are not significantly different from other languages. Like the inspector, debugging should be in high level. Surprisingly, some object-oriented language implementations fail to provide source code debugging, and depend on the programmer's ability to trace through generated code. The debugger should allow the user to inspect any object, modify source code, and resume execution.

Extending the Environment

Although we'd all like to have a perfect programming environment, its unlikely that any single system will meet everyone's needs. Ideally, a programming environment should be extensible so that the programmer can customize it and add new tools. Although this sort of architecture is not common, some interactive systems, including the Common Lisp Object System (CLOS), Actor, and Smalltalk, are completely open ended. This makes it possible to develop additional tools that you can seamlessly integrate. If you need a special kind of browser or inspector, you can add it to your system without abandoning the existing tools.

Selecting a Programming Environment

Although object-oriented programming languages can increase productivity and reduce development costs, they are only part of the software development process. A programming environment is more than just a language translator. It consists of all of the tools available to the programmer to support the development process.

When selecting a programming environment one should consider the following questions.

- What predefined classes does the system include?
- Can you browse the existing classes and examine the protocol for objects?
- Does the system provide rapid turnaround, enabling interactive testing and experimentation?
- Can one inspect objects in high-level terms and modify their values without writing additional debugging code?
- Does the environment support source-code debugging?
- Can you modify the code and inspect objects from within the debugger?
- Can you customize and extend the programming environment with new tools?

Although no programming environment is perfect, the answers to these questions can help you determine how the environment will affect your ability to deliver finished applications on time and under budget.

Zack Urlocker *is manager of Developer Relations at The Whitewater Group, the creator of Actor, an object-oriented language for Microsoft Windows.*

From Languages to Environments

by Zack Urlocker

This month's column explores programming environments, their relationship to object-oriented development, and some criteria you can use to help select a productive environment.

With all of the emphasis on object-oriented programming languages, we sometimes forget the importance of the programming environment. After all, the language translator is only one of the tools available to the programmer. Other tools that make up the programming environment include editors, inspectors, debuggers, design tools, and so on.

Interestingly enough, there's always been a strong tie between object-oriented languages and programming environments. Much of the early research in programming environments was done at Xerox PARC and was incorporated into Smalltalk-80 and Interlisp-D. Smalltalk remains one of the best known examples of a programming environment.

The object-oriented programming paradigm has a broader focus than traditional procedural programming. The goal of object-oriented programming is not only to create an efficient robust system, but to do so by reusing existing code. As a result, object-oriented programming requires more than just a programming language.

Beyond Language

As Adele Goldberg, president and CEO of ParcPlace systems, and formerly of Xerox PARC, has said, "object-oriented programming is not enough." Organizations trying to take advantage of OOP must consider not only the programming language, but more importantly, the design methodologies, class libraries, and programming environments they will be using. Selecting a language without regard for the programming environment is like buying a house without considering the location.

Although classical structured design methodologies dictate that a system is completely specified and designed before a single line of code is written, many complex projects benefit from an exploratory approach to programming. Object-oriented programming systems such as Smalltalk and Actor are often used to develop prototypes of applications that evolve into a production quality system. What is it that makes these systems suitable for exploratory programming?

In order truly to benefit from the reusability of object-oriented programming, one needs a class library from which to draw. The richer the class library, the bigger the headstart one has in application development, thus, the easier it is to prototype a system. Although most object-oriented languages provide at least a basic class library from which to start, others such as C++, are notable in their total lack of a class library.

You've Got to Find It to Reuse It

A class library increases the ability of programmers to reuse existing code. However, in order to reuse code, one has to be able to find it. Thus, there's an increased requirement to view classes and become familiar with how they're used. Although one doesn't need to see the source code of how the class is implemented, one does need to know the class's purpose and its *protocol* — that is, the messages its objects understand.

Code browsers provide a convenient mechanism for viewing existing classes and creating new ones. A good browser goes beyond text editing to allow you to understand the classes and see how they're used. In Actor, for example, the browser shows the classes in hierarchical order with their components, or instance variables, and methods. One can also find the senders and implementers of any message, giving quick cross-reference capabilities.

Benefits of an Interactive Environment

Of course, becoming familiar with the class library is only part of object-oriented programming. You also need support for writing, testing, and debugging your code. An interactive environment can go a long way toward improving programmer productivity in any language. It is especially valuable in an object-oriented system in which you want to reuse existing code.

As you create new classes that inherit behavior from existing ones, there is a need to test your work. An interactive environment allows programmers to eliminate the tedious edit-compile-link-test cycle of traditional programming to provide faster turnaround time and more direct feedback. The programmer should be able to edit code, compile it, and test it within about three seconds. A time longer than three seconds will interrupt the programmer's train of thought and increase the cost of testing.

did not have a booth, they had a strong showing with their system running in a suite. It was surprising to see large placards in the New Orleans airport touting the Object Design product, ObjectStore. Onotologic was showing their ONTOS product, a reincarnation of V-BASE in C++. Symbolics was showing their Statice OODB product as well as other offerings. Object-Sciences Corp. also did not have a booth but was introducing folks to their OBJECT-BASE OODB product in their suite.

The hardware vendors were not showing hardware to sell but rather software packages that they have available on their platforms. Apple was showing their new C++ implementation and other OO tools. AT&T was pushing C++. Digital was showing Trellis-Owl, still not sure whether or not they would sell it. HPwas showing OO languages and tools on HP and Apollo. Some of their analysis tools in internal development are very interesting. International Meta Systems and Smalltalk Express were showing RISC based Smalltalk accelerators.

The number of training and consulting firms is growing, the emphasis is on C++ training. Knowledge Systems announced their Apprentice Program, an unusual, full-immersion approach to training.

The bookseller and magazine vendors are also reflecting the growing interest in object-oriented programming with many titles coming out in the next few months. We note with pride that HOOT was introduced at OOPSLA with a very positive response.

Summary

What does it all mean?

Object-Oriented Technology is there to stay. There was a new level of maturity at OOPSLA '89. While the language wars (C++ vs. Smalltalk vs. Eiffel vs. Object-C vs etc.) continued, they were considerably muted. Much work remains to be done, especially in the areas of analysis and design. The vendors are learning to use the technology to solve customer problems, not to just sell the technology. We have just begun a long journey.

We'll see you next year in Ottawa.

C'ing into the Future

by Keith W. Howard

"So what does a 500-pound gorilla do on its birthday?" asks Peter Deutsch, creator of Smalltalk, the original object-oriented programming language. Answer? "Anything it darn well pleases." The gorilla in this story is a metaphor for C++. He shared his tale with an audience of one thousand during a panel discussion at the recent OOPSLA Conference in New Orleans . On the panel with him were Bjarne Stroustrup,"father" of C++, and Gregor Kiczales of Xerox PARC, designer of CLOS.

Like much humor, the above story has its dark side, and it is not especially complimentary. The subscript is saying that despite the shortcomings of C++, despite some cryptic syntax, and despite the flaky support for dynamic binding of functions — a litmus test of an object-oriented language — C++ will be a success and most likely, a huge success.

One of the primary mandates of C++ was that it should be upward compatible with C. Therein lies its assured success. The wisdom of protecting one's installed base by offering incremental enhancements to an already existing environment was not wasted on AT&T or on Mr. Stroustrup. The strategy makes brilliant marketing sense and reasonably fair technical sense. After all, one cannot expect a user just to dump his entire programming investment, which has taken years to make for some newfangled language.

Have you heard this story somewhere before? I have too. I don't recall the last reported count of the number of lines of COBOL that exist in the world

today, but like the sales of McDonald's hamburgers, it's millions and millions and growing. If C++ is the 500-pound gorilla then COBOL is the 5000-pound elephant and a friendly elephant at that. Strong and sure footed, it has been pulling its weight in the area of business applications for three decades with hardly a groan of complaint. The groans come from chronic C addicts.

Mention the probability of Object-Oriented COBOL to software engineers and they may be seized with a fit of hysterical laughter. This is a curious and not uncommon phenomena that it is worthy of a brief examination. At its core is a basic human desire to sense purpose in one's life. Not just another body on the planet. The opportunity to feel purpose is open to us through creativity. In the world of exploratory computer software, this translates into some new product or practice that claims to offer a finer or more efficient use of one'sr computer, which use no one has thought of before.

Having slaved day and night to produce a masterpiece, it comes as a shock to be asked to articulate one's creation in the programming language of accountants and bookkeepers — COBOL — the programming language of the common people. Once articulated in a syntax or a descriptive paper understandable to all: one lays oneself open to the most dreaded enquiry of all.. "Is that all there is to it?" In one ghastly moment the whole shining edifice of one's creativity can come crashing down.

A more open frame of mind can foil such a moment. Wouldn't you wish for as many people as possible to benefit from your inventive genius? Of course you would. So don't hide your light beneath a bushel and don't hide your sparkling new computing capability beneath arcane syntax or obtuse technical writing.

Love it or hate it, the enormous installed base of COBOL means it will not simply vanish, so you'd better make friends with it if you have plans to penetrate the huge marketplace of commercial and business computing. This was exactly the point Peter Deutsch was making about C++. In the absence of a quantum leap in language technology, the force of inertia to stay in the comfort zone of the familiar will remain strong.

COBOL compilers are available on a broad range of computers and workstations. They run under mainframe operating systems, UNIX, PC DOS and OS/2. True, COBOL is showing its age in places. It still carries a great deal of baggage from its punched card days and it will need work to allow it to support an OO feature set. But the language is far from stagnant. It will not be long before we see Object-Oriented features in the COBOL language. Hewlett-Packard has already submitted a paper to the COBOL CODASYL committee laying out some ideas for syntax to define classes, constructors, destructors, and their usage. Other vendors are expected to do the same.

I see no reason why advanced OO features should not eventually be offered under a COBOL syntax, such as dynamically alterable object formats and the "slot" like capabilities found in Stanford's SELF language. What would a commercial shop do with all this power? The development of commercial and financial class libraries could bring dramatic advancements in services through common data structures across institutions. Don't forget that one can do more in COBOL than just write programs to balance one's checkbook. It is another syntax, another tool. Give the tools to the people and it is remarkable how creative many will be.

A more immediate need is for good syntactic support for GUI interaction to products like Presentation Manager, MOTIF, and Open Look, all of which operate in an OO paradigm. It would not be a bad idea for vendors of Object Oriented databases to start thinking about friendly COBOL gateways to their products, even if this means a preprocessor that translates COBOL syntax into their DML.

The introduction of OO systems and strategies into the world of business computing will stand a greater chance of success by taking a tip from the gorilla.

The thought of OO functionality being available to the COBOL community still causes some snickering among hard nosed software engineers. They overlook the fact that the OO paradigm is describable in many languages. Bjarne Stroustrup offers a mature viewpoint to such shortsightedness. He recalls the time when he was asked to speak to a Simula users group about Object Oriented programming. He asked the conference organizer if it would not be somewhat tactless for a C++ expert to be talking to a group of Simula programmers. "Oh no," was the response, "we're not hung up on syntax."

Keith Howard *is an 18 year veteran of the computer industry and has worked as a consultant in the United States and Europe. He lives in San Francisco.*

Object-Oriented Environment in C++: A User-Friendly Interface

by David Hu
(MIS Press, 1989)

"If you can't say anything nice, don't say anything at all," my mother told me. If I followed my mother's advice with regard to *Object-Oriented Environment in C++: A User-Friendly Interface*, by David Hu, this would be a very short book review.

I had difficulty in reviewing this book because the book does not present what its title implies. Section 1 purports to introduce the Fundamentals of Object-Oriented Programming. Unfortunately, the example used, an automobile parts inventory system, fails to make a clear distinction between class hierarchy and class composition. The class "Fuel System" does not "inherit" any useful behavior from class "Sedan." That is composition, not inheritance. This is not the first book that confuses inheritance with composition, and it probably won't be the last! At the end of the first section, I found myself asking, "What does this have to do with object-oriented programming? It looks like an ordinary inventory application to me."

The author proceeds to compare OOP with conventional programming. He does this by discussing Smalltalk for two paragraphs and then launching into a ten page example of how to paint a Chinese character using C++. The exhaustive example fails to enlighten the reader as to the "fundamentals" of object-oriented programming.

Chapter 2 is entitled "Object-Oriented Programming in Smalltalk." It uses Smalltalk V/286 as the basis of the description. The next fifty pages are devoted to a description of the classes provided by Smalltalk V/286. To understand this chapter, it would be best to have the entire Digitalk product, since the examples presume that the reader is using the Class Hierarchy Browser while reading this text. If one had the software, one would also have the excellent tutorial that comes with it, and would not need this book. Much of the content of this chapter appears to be taken from the Digitalk document.

One particular feature in the Smalltalk section is a questionable contribution by the author, which is the introduction of the metasymbol "=>" to indicate that the object to the right of the metasymbol is the receiver from the expression on the left. This nomenclature is never defined anywhere in the text, leading the reader to believe that "=>" is a part of the Smalltalk syntax, which is not true.

The chapter on "Object-Oriented Programming in C++" borrows heavily from the Stroustrup book (*The C++ Programming Language*, Addison-Wesley, 1986), as the chapter on Objective-C borrows heavily from Brad Cox's writings (*Object-Oriented Programming: An Evolutionary Approach*, Addison-Wesley, 1986). If the reader wants to know about these languages, he can go to the original source.

Section III finally gets to the topic of user interface design. In Chapter 5, the implementation of class TextWindow in C++ consumes the next 30 or more pages, padded out by long, multipage listings of source code. The book claims that the source code is available on diskette, which would be required if one is going to put this to any serious use. I was unable to acquire the source diskette in time for this review, so I am unable to verify any of the examples. The class TextWindow implemented has a variety of operations for painting text in a window on the video screen, but does not support any input operations other than a crude copy from keyboard to screen. The example falls far short of developing a class that would be useful for implementing an interactive dialog with the user, which is an important component of user interface design.

The above complaint is partially addressed in Chapter 6, in which is implemented a set of graphical controls, implemented atop the Flash Graphics package available with the Zortech C++ compiler. Again, the reader is deluged with pages of sample code, interspersed with terse descriptions. Without the sample code, this book would be about one third the size of its rather hefty 557 pages.

Subsequent chapters go on to discuss expert systems and C++, although one wonders how this fits into the original goal of the book. The book did have an original goal, didn't it?

Needless to say, I don't recommend this book. If the book had been billed as a case study, for example, of using object-oriented techniques in an industrial application, one could excuse the digressions into detail. Billed as an "Object-oriented environment in C++," it fails miserably.

Reviewed by **Ken Burgett** *of Callosum, which specializes in object-oriented programming in Smalltalk and C++. (408) 258-5113*

sionally it will even be necessary to allocate a single method across processor boundaries. The same applies for the data that any object encapsulates.

10. Establish the implementation software architecture: the languages, utilities, and services which will be available in the target environment. As practical, mold the environment to the problem, rather than the problem to the environment. Although desirable, it is not absolutely necessary to use one of the strictly object-oriented languages. *Synthesis* allows for mapping an object-oriented functional model into a conventional language. Most of the quality of the analysis will be preserved. Good results are obtainable with COBOL and FORTRAN.
11. Package the objects allocated to each processor. Declare the package(s) (which could be programs, tasks, objects, interrupt handlers), which will run on each target machine. The nature of the packages will depend on the language and operating environment.
12. Design the internal aspect of each of the components that have been declared. In many cases this step is a variation on traditional hierarchical design. The hierarchies that are designed for environments where inheritance is available (or effectively simulated) will be different from hierarchies that are designed for environments without inheritance. Data encapsulation should be well represented in either case.
13. Code the system, probably using an incremental construction strategy. If you use a very radical approach, you may begin coding quite early in your project. Also, you don't have to wait until the last moment to start coding certain utility and structural object classes.

Summary

Software engineering is a growing and evolving discipline driven by increasingly complex technology and by more and more sophisticated user problems that we have to solve. As the current methods become less adequate to the task, new methods, such as object-oriented ones, will supersede existing methods in forward-looking companies.

The authors may be contacted at Wayland Systems Inc. in Seattle at (206) 525-0525. They encourage your feedback, comments, and tales of experiences with object-oriented system development.

© Copyright 1989 Wayland Systems, Inc., Seattle WA. All rights reserved.

Company Showcase: An Information Service

Regularly appearing in HOOT, this section introduces you to companies offering various products and services in the object-oriented field. Every company which sells an object-oriented product or service (old or new) will eventually appear in the Company Showcase.

Included here is a brief description of the company and its product or service, atributes, with name and phone listing for further contact. This section is provided as an information service to readers and no endorsements by HOOT are implied. Summaries are provided by the companies. Companies wishing to be included in the Company Showcase should send appropriate information to Jeff McKenna, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062. Tel. (503) 692-1889

Rogue Wave

Rogue Wave is located in Seattle, Washington, and specializes in object-oriented numerics.

The principal partners are Drs. Thomas Keffer and Rick Romea. Both are experienced applied mathematicians and programmers, with strong backgrounds in numerical modeling of fluids and linear and nonlinear optics, including lasers.

Rogue Wave offers a set of high-level libraries and objects designed to make it easier to build and maintain large numerical projects such as models, simulators, data analysis routines, and image processors. They are designed to compile under the MS-DOS and UNIX operating systems.

Current projects include development of enhanced C++ classes, including graphical interfaces and monitoring systems, and the development of object-oriented analysis and modeling tools, such as a workbench for analysis of coherent propagation and graphical interfaces to databases.

For more information, contact:
Dr. Thomas Keffer; or
Dr. Rick Romea
Rogue Wave
P.O. Box 85341
Seattle, WA 98145-1341
(206) 523-5831

TurboPower

TurboPower Software is a leading developer of products that improve the productivity of Pascal programmers. Founded in 1985, TurboPower has developed a series of acclaimed products including T-Debug Plus, a symbolic run-time debugger for Turbo Pascal 3.0 and 4.0, winner of a PC Magazine Editor's Choice.

Current products include Object Professional 1.0, a library for Turbo Pascal 5.5, which includes over 100 object types for user interfaces, data objects, and system oriented routines.

For more information, contact:
Robert Benson
Director of Marketing
TurboPower Software
P.O. Box 66747
Scotts Valley, CA 95006-0747
(408) 438-8608
Fax (408) 438-8610

O-O Technology In The News

Recently, we started getting regular mail from a commercial clipping service (US only). We just specified the words 'object-oriented' and we waiting with some interest as to what we would see. We had tried this some years back and didn't get enough to do much of anything.

We were very impressed. In the last six weeks we received some 100 articles from the weekies and some 65 articles in the monthlies as well as a few from dailies across the nation. About half of all the articles were related to products, announcements and reviews. some 40% was general information about object-oriented technology: What it is or How to program in it. The remainder was general announcements (OMG for example) and use of the term as a marketing pitch (Of course we're object-oriented!).

We will be tracing this for a while and will give you some indication of the growth (or decline!) of the number and distribution of the articles.

In the meantime, here are some of the more notable ones:

Researchers involved in a 15 year, $3 billion effort , The Human Genome Project , are using object-oriented technology to decipher our genetic makeup. Both OO programming and OO data bases are being used in the massive undertaking. The type and kind of data being recorded is changing quite rapidly and the flexibility of OO techniques is essential. (*The Scientist*, September 4, 1989)

John Davis of Anderson Consulting gave a brief and clear description of the process of discovering , use, and refining of objects in the context of business software His statements are the clearest we have seen on how OO development really proceeds. (*I/S Analyzer*, September, 1989)

OO technology has matured enough to be able to laugh at ourselves. Scott Robert Ladd wrote a very amusing description of an object-oriented programming conference (anyone we know?) He manages to bring up some very pertinent points at the same time. (*Computer Language*, October, 1989)

Computer Language has been going at OOT hot and heavy. As part of their effort, articles by Ed Yourdon and P.J. Plauger in the October, 1989, issue focus on some of the unresolved issues of OOT. Yourdon warns us that engineering is required and Plauger cautions against uncritical acceptance of new ideas.

James Martin, the long-time industry guru, recently had a four part series on object-oriented techniques (*P.C. Week*, September 4, 11, 18, and 25, 1989) In the articles, he introduces the concepts, describes OO programming, gives some example applications, and discusses OO data bases. In the first article he gives a good description of current uses of the word 'object': ". . . in most cases it encompasses both properties (data) and behaviors (procedures). It is used loosely to describe the icons in a user interface, the entities in databases, and the elements manipulated by system software in newer machines. In programming languages, it refers to the encapsulated units of properties and behaviors."

DEC has described the use of possibly the worlds largest application of OODB technology. It was used to hold the design database of a six year project at DEC to develop a new style of interconnect technology and a new mainframe. "The object-oriented design database used in creating the tools and designing the mainframe eventually occupied 5,847 Gbytes in more than 18,000 files." (*EE Times*, October 2, 1989)

Calendar

The HOOT calendar presents conferences and meetings that are focused exclusively on object-oriented technology. It should be noted that many other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to the Editor, HOOT, 8825 SW. Umatilla St., Tualatin, OR 97062

DECEMBER 3-6, 1989

First International Conference on Object-Oriented and Deductive Databases—Kyoto, Japan. Sponsored by the Information Processing Society of Japan and the Advanced Software Technology/Mechatronics Research Institute of Kyoto. Features tutorials, technical sessions and exhibits. Contact: Won Kim at MCC (512) 338–3439

DECEMBER 4-8, 1989

International Workshop on Software Engineering and Its Applications— Toulouse, France. Contact: Jean-Claude Rault, EC2 269-287 Rue De la Garenne, 92000 Nanterre, France. Tel. 011–331–478070–00

DECEMBER 5-7, 1989

The Object-Oriented Symposium— Presented by Digital Consulting, Inc. and chaired by Larry L. Constantine, in Chicago. Repeat of the symposium presented in October 1989.

JANUARY 22-24, 1990

SCOOP-West—Techmart Center, Santa Clara, California. Co-sponsored by JOOP and the Wang Institute of Boston University. Contact: Bob Daniels, Wang Institute of Boston University, 72 Tyng Road, Tyngsboro, MA 01879, (508) 649-9731.

Standards for OO-DBMS: A Manager's Guide

by Tom Atwood

Object-oriented technology is rapidly gaining support as the technology of choice for nonrecord oriented applications. Its popularity comes at a time when the computing world has become highly standards conscious. Yet, many people suggest that object-oriented technology is too nascent for standards, and that standards would inhibit the innovation and creativity necessary for object-oriented software to mature. Others suggest that without standards, it will be impossible to commercialize object-oriented technologies. In fact, like most black and white positions, neither is true. In this article I examine how standards are created, where they are needed for object-oriented software, and what you, the buyer, should consider when making an investment in object-oriented software.

The Drive for Open Systems

Standards don't emerge by some orderly, well-defined process. It is the users and the marketplace that creates standards. As Esther Dyson of the newsletter Release 1.0 puts it, "the marketplace votes and de facto standards are set." Only later are these de facto standards formalized through a variety of national and international standards organizations.

Historically, a few companies, such as IBM and Digital, have had the market power to establish standards by fiat. But this is becoming more difficult as users demand open systems. IBM's trials with OS/2 and microchannel bear witness to the power of the user community.

The drive for open systems is fueled by the economic necessity for users to leverage their existing investment in computing hardware, software, and operating systems. For purchasers and builders of object-oriented database management systems, there are two clear messages:

1. The DBMS must run in an 'open' software environment
2. The DBMS must itself embody a set of Data Model and DML standards.

How the latter will evolve is the topic of this article. I comment on the first only to the extent of suggesting that a viable OO-DBMS must support Unix and the other mainstream operating systems such as VMS, Macintosh, and OS/2. It must be able to run in a distributed LAN configuration using widely supported networking protocols. The interactive interfaces and utilities should be X-window based, and adopt the look and feel of the system on which they are being run: OSF/Motif, Open Look, or PM. Finally the DBMS must interface to the major object-oriented languages. C++, the object-oriented superset of C, is emerging as the dominant object-oriented language. The current C++ installed base numbers nearly 50,000. There are projections that it will jump to over 200,000 by year-end 1990 when Microsoft and Borland begin shipping compilers. The current AT&T release of CFRONT 2.0 has significantly improved the stability and viability of the language for developing commercial systems. AT&T has laid out a long-term development plan for C++, which includes support for parameterized types and exception handling. AT&T also recently endorsed formation of an ANSI standard committee for the language.

Object-orientation is also becoming prevalent with other commercial languages such as Pascal, Basic, LISP, and ADA, all having object-oriented derivatives. Finally, that bastion of the MIS world, Cobol might jump on the OO bandwagon. Megan Adams and Dmitry Lenkov of Hewlett-Packard in Cupertino has submitted an Object-Oriented Cobol Working Paper to the CODASYL Cobol Committee that develops recommendations for the ANSI Cobol standard.

The Need for Object-Oriented Data Management

This rapid movement towards the development of mainstream commercial object-oriented languages is creating a need for object-oriented database management systems that support this style of programming directly. It is simply too expensive for the programmer to have to write code which maps the in-memory representation of objects that he wants to store permanently to an on-disk format, and then code to read that data from disk and reconstruct the in-memory representation of the objects when he wants to use them

structured techniques and those with object-oriented development complement each other. The best aspects of structured techniques could work wonders for object-oriented development, and the major principles of object-oriented philosophy could solve the practical problems of structured techniques.

Thus we came up with *Synthesis*. *Synthesis* is an Object-Oriented Analysis and Design Method that synthesizes that which works works well in structured analysis and design with the important characteristics of object-orientation.

If you currently use structured analysis and design, then to use *Synthesis* you need not discard your previous expertise and training investment. The *Synthesis* object-oriented organizational principles will enhance the development and organization of your structured analysis and design models. However, you don't have to use an object-oriented programming language (although we strongly encourage it) if you choose to use *Synthesis* for analysis and design.

If you currently use object-oriented programming, *Synthesis* provides an analysis and design method as a front-end to programming, with which you can elicit user requirements and develop a software structure tailored towards maintainability and reusability.

Below we outline the specific steps of *Synthesis*.

Overview of *Synthesis*

Encapsulation is one of the fundamental principles driving object orientation. It focuses neither on process nor on data, but on their interrelationships. A method such as *Synthesis*, which is designed for an object-oriented environment, must therefore be simultaneously directed towards both process and data. The modeling focal point is an information model containing entity and relationship types (from which objects are derived), augmented with the relevant processes and states. System information that was formerly distributed throughout a traditional model is now organized around the object classes derived (mainly) from entity-types and relationship-types of the information model.

The steps through which you drive a *Synthesis* Model are:

Analysis

In parallel build:

1. An Augmented Information Model for the application environment. At minimum, include entity and relationship specifications, policies regarding creation, retrieval, updating and deletion of each, entity and relationship state transition diagrams. Include an entity type for each terminator on the Context Diagram.
2. An Events Dictionary for the application. The Events Dictionary contains stimuli, responses, constraints, event recognition mechanisms, and the like. Its content is an abstract of the content of the system. Nouns in the Events Dictionary will suggest entity types for the Augmented Information Model. Verbs will suggest relationship types.
3. A context diagram for the application.

When completed, these documents form the basis for an object oriented essential model. To continue:

4. Declare an object class for each entity and relationship type seen in the Augmented Information Model. Some of the event types will also result in the declaration of an object. Organize into a class hierarchy to show inheritance. Declare one object, representing the system, at the top of the class hierarchy.
5. Populate each object with its initial methods, attributes and STD as needed. The initial methods for each object generally include one for creation, a set for retrieving and updating attributes and one to eliminate gracefully the object when it is no longer necessary.
6. Build a Neighborhood Diagram for each event in the Events Dictionary. The Diagram shows all of the object classes and messages which collectively process the topic event. For each event, identify the event recognizer, event manager, and methods for event resolution. New objects may be introduced for some of these roles. Specify methods for each object class. Support with a data dictionary for class variables, instance variables, and messages.
7. Transform Neighborhood Diagrams into a leveled set for presentation. A substantial portion of the organizational information provided by a traditionally-leveled model is now provided by the class hierarchy, which is a form of leveled set. In a manual modeling environment, leveling a Synthesis model is of debatable value. We do see value, however in a CASE supported environment where the leveled view is tool-mediated.
8. Complete the model with the various supporting texts: method specifications, message dictionary, various appendices, and a glossary as needed.

Design

Design follows analysis; the statement of implementation follows the statement of required function.

9. Assign each object class to a processor. Sometimes it will be necessary to allocate an object's methods across processor boundaries. Occa-

Synthesis: An Object-Oriented Analysis and Design Method

by Meilir Page-Jones and Steven Weiss

"We don't need a course on analysis; we're object-oriented."

This calamitous *non sequitur* began a project debacle, whose resulting "system" (to use the term loosely) ignored a major portion of the users' needs. Half a dozen programmers swept across the programming plain, strewing object classes all about them. These classes — over 1,000 of them — had ill thought-out properties, redundancy, and gaps in their provided features. The more things change in systems development, the more they stay the same!

How can we avoid such object-disorientation on future projects? In the OO world are we doomed to inherit some of the undesirable features that we've grown to know and hate in traditional software development? And where — in the words of many of our clients — do object classes come from? Does one just guess them?

Another crucial issue in the development of object-oriented systems is the arrangement of responsibilities among object classes. This too has caused much confusion among our clients. The question is best couched in terms of the famous "object-oriented farm" problem. Let's say that on a farm a cow needs to be milked. Do we send a message to the cow, saying "Milk yourself" ? Or do we send a message to the milk, saying "Uncow yourself"? There has been no widely accepted criterion in object-oriented development to distinguish between these possibilities.

We believe that this Disastrous Object-Oriented Muddling is avoidable. The problem is that object-orientation has been widely touted as a revolutionary approach — a complete break with the past. This would be fascinating if it were true, but it isn't true. Like most developments in any engineering discipline, the object-oriented approach is a refinement of some of the best software-engineering ideas of the past.

The key ideas of encapsulation and implementation-hiding (often misnamed information-hiding) were proposed almost 20 years ago by folks such as Larry Constantine and David Parnas. Rudimentary forms of message-passing to an implementation-hiding structure were proposed by Barbara Liskov, Glenford Myers, and Meilir Page-Jones in the 1970s — as well as by the originators of the Smalltalk language.

Perhaps the most revolutionary developments in object-orientation are: 1) the use of the class as a central management facility for a group of instances; and 2) the provision for one class to inherit features attached to another class. Although one could regard these developments as being offshoots of the notion of abstract data types and subtypes, only object-oriented techniques have raised these developments to useful heights.

Although object-oriented systems development has exciting advantages, there exist the following concerns:

- Object-orientation has been treated as a programming-language issue.
- Many programmers writing object-oriented code are not familiar with the best current analysis and design techniques.
- Object-orientation is treated (as discussed above) as a break from the past.
- The myth has arisen that object-oriented programming can take place successfully without benefit of analysis and design.

Before we start to sound too critical of object oriented systems development, we should mention that neither is everything perfect with current structured techniques. For example, in structured analysis:

- Event-partitioning works well, but leaves stored data "out in the cold" with no clear way to enforce its invariant properties or to provide any implementation-hiding capabilities.
- Transitions among valid states of an attribute are effected by various event-partitioned processes, rather than being managed in a single portion of a model. Alternatively , if one chooses to use so-called real-time notation ("dotted bubbles"), then the management of state transitions is in one part of the model, but is separated from other stored attributes of the same entity type.
- Techniques for integrating the information model of structured analysis with the process model have never been completely clear. For example, there is no way to exploit information-model subtypes in the process model with a technique such as inheritance.
- The data dictionary of structured analysis makes no distinction between data instance and data type (and far less data class). Thus there is no way to model commonality of properties across data instances.

These are some of the main drawbacks in structured analysis as it is currently observed. They are not devastating drawbacks, but they can be extremely annoying in practice. We realized that the problems with

again. Early experiments by Atkinson's group in England suggested that eliminating this explicit, programmer-built mapping code, could shrink the size of many applications by as much as two-thirds. Given this clear benefit, the question arises as to what standards are needed and how they will emerge as OO-DBMSs become mainstream products.

OO-DBMS Standards

Crystal ball firmly planted on the table before me, I predict that standards for object-oriented data management will emerge in 3 areas:

1. Data model
2. Data manipulation languages
3. Distributed object transport protocol

Data Model Standards

Some consensus on the data model exists today in the object-oriented community and this is the basis for the development of the next generation of commercial object-oriented DBMSs. Specifically, objects are instances of types (or 'classes'). Types define both an abstract specification (or 'public' interface), which instances of the type obey, and an implementation ('private' part).

The abstract specification defines the state and behavior of instances of the type. State is captured as attributes for which instances of the type carry values and relationships in which instances of the type may participate. Behavior is captured as a set of operation signatures. Each operation signature contains the name and type of the arguments to a given operation, the type of the returned objects, and any exceptions that can be raised by an invocation of the operation. The implementation consists of a set of data structures which serve as the representation of the type, and a set of methods, which implement the abstract behavior specified by the operations declared in the specification in terms of the underlying representation. Neither the representation nor the methods are visible outside the type definition. Other objects interact with objects of this type only through the operations, attributes, and relationships defined in the specification.

Variations between models come in a few characteristic areas:

1. How state is captured in the specification
2. Whether attributes, relationships and operations are treated as full fledged types
3. What type of 'inheritance' is supported between types
4. Whether persistence is orthogonal to type

This latter issue deserves special comment. Historically, database management systems and programming languages each defined their own independent type systems. All instances of the types defined in the database schema were persistent. All instances of types defined in the programming language were transient, that is, they did not outlive the program that created them. A much simpler and cleaner solution is suggested by Atkinson's group in England, which is to treat the lifetime of an object as an orthogonal issue to its type, have a single unified type system, and allow individual instances of these types to be transient or persistent. This is such a clear simplification that it seems destined to be part of any OO-DBMS standard.

What I expect to see emerge on the data model question is a set of level-1, level-2 standards, much as in the early Codasyl DBMS proposals. A viable level-1 proposal might include only objects as full fledged types (not operations or properties), allow state (attributes and relationships) to be defined on object types as well as operations, support multiple inheritance, and define persistence as orthogonal to type.

Data Manipulation Language Standards

In the relational era, a single DML sublanguage, SQL, was proposed both as an interface for interactive users, and as a sublanguage to be embedded in programming languages. Experience has revealed that by trying to do both, SQL was ideal for neither job. Today, SQL is not generally used by the nontechnical interactive user to generate queries, rather, graphical or by-example interfaces have been defined which generate SQL as their intermediate representation. I believe the OO generation will move directly to this latter solution for interactive query.

The OO generation may take a different tack on embedded sublanguages also. The relational era defined one DML, SQL, and forced it, however awkward the fit, into several distinct programming languages: COBOL, PL1, FORTRAN, C, and ADA as examples. The rationale for this was that a programmer going from COBOL in the morning to PL1 in the afternoon would at least be able to use the same DML sublanguage. Since the syntactic forms of any two programming languages were different, however, SQL had to be a fairly complete language unto itself. It had its own definition of expressions, statements, and variables. And since they were designed to fit into any language, they clashed with the definition of expressions, statement, and even sometimes variables in the base programming language.

In the OO world, the rich data model blurs the distinction between the language and the database and will lead to a different approach for defining DMLs. The

data model will define a common semantic and then to map that semantic into each specific programming language, using syntactic forms that fit smoothly with each distinct language. So what we will see is not a single DML standard, but a family of standards, a standard DML for each host language. Examples of this are emerging today. AT&T and Object Design have developed very similar DML proposals for C++. ServioLogic has developed a DML for Smalltalk. Symbolics has developed a DML for LISP. HP has submitted to the ANSI X3.J13 committee a proposed DML binding for COBOL.

Distributed Object Transport Protocol Standard

OO systems are entering the market in an era of local-area-networks, connecting workstations and PCs to servers. Although these machines often have different architectures, users want to mix machines from different vendors and the pressure for well-defined object interchange standards will emerge sooner in the OO era than it did in the relational era. Further, because the bulk of compute cycles are in the workstations or PCs rather than in the server, there is likely to be pressure for moving either the data to the site of execution of the operation, or conversely, for moving the operation to the data. This is possible because the operations defined on object types have the full generality of programmed procedures in OO systems,and are not limited to combinations of join/project/select operations defined on tuples as in the relational world.

Some of the OO systems entering the market next year will give the programmer the choice between moving the data to the workstation or specifically requesting execution of the operation on the server using an RPC call. Over time, I would expect to see standards emerge in object transport protocols in which the system may decide where to execute a particular operation, based on dynamic criteria such as availability of resources, rather than forcing the programmer to explicitly specify the site of execution. (ed: Apollo systems do this today for some types of operations.)

Parallels to Relational Era

It took nearly ten years for a standard DML to emerge in the relational era. The timeline below shows the steps between first publication of SQL working papers in 1979 and the final emergence of an ANSI standard 7 years later in 1986. Even today, it is not clear how much real portability the existence of the standard confers. All of the major relational DBMS implementations are supersets of ANSI SQL and, to that extent, are incompatible.

1975 to 1979 . . . IBM publishes SQL work in research journals

1980 . . . Oracle ships

1981 . . . RTI ships Ingres

1982 . . . ANSI sub-committee agrees to evaluate relational standard. IBM offers to put SQL in the public domain. Committee opts to develop an independent standard, RDL

1983 . . . IBM ships DB2

1984 . . . ANSI committee abandons RDL, bowing to market pressure for SQL

1986 . . . August, first ANSI SQL standard published

Possibly the clearest message of this history is that markets drive standards, not the other way around. The original ANSI standards committee work attempted to develop a better relational DML than SQL, but as SQL became the de facto standard in the market, the committee shifted to endorse SQL. I asked Don Deutsch, Chairman of the X3H2 Committee on Databases what advice he could give those of us in the object-oriented generation. He commented: "No standard is created from scratch. Having something that is in use in the marketplace is always the best way to get standards efforts moving. Also vendors should talk to each other." This latter process has already begun with the formation of Object Management Group (OMG) last spring.

OMG has formed an Object Database Interface Task Force headed by Dave Hentchel of Prime/CV in Natick, MA. Dave reports that the task force has defined its mission as "providing an implementation for the persistent storage of objects, agreement on a data model, access from a variety of languages and interpretability with existing databases." Thus the foundation for establishing standards has already begun. Let me make a few comments on the importance of interoperatibility.

Interfacing to Existing Relational Databases — Commercial Reality

Databases, once implemented in a given technology, are rarely replaced. Even if new applications are written using object-oriented programming and OO-DBMSs, pragmatically it is necessary for the new applications to have access to data stored in existing relational databases, or even older IMS databases. There are two options for this: (1) force the programmer to interrogate explicitly both databases, using SQL for one, and OQL for the other, and himself handle the merging of record-oriented data with object-oriented data, or (2) allow the programmer to issue an OQL request to the OO-DBMS, and have the OO-DBMS map that into an SQL request when the necessary data is available only in a pre-existing relational DBMS.

The second option is clearly the long-term solution of choice. It allows new programs to be written at the higher level of abstraction implicit in the OO model, but retains access to data still stored in older, lower-level relational DBMS. It allows an easy migration path in which new applications or new functionality can be introduced to existing systems using the OO paradigm, without requiring the wholesale conversion of the existing database and existing body of non-OO applications that use or maintain it.

Because of the dramatic performance advantage of OO systems for nonrecord-oriented data, new applications written to access graphics, voice, text, etc. will be written to run directly from a OO-DBMS. But many of these same applications will have to access existing bodies of relational data. This is distinct from simply adding an OO interface on top of an existing relational DBMS. The latter will not be fast enough for most of the new multimedia applications that the true OO databases serve.

What should the Prudent Manager Do?

The tension between the need for innovation and the need for standards to foster commercialization has always made the job of management in high technology a challenge. This is especially true in that object-orientation promises early adaptors a competitive advantage through lower development cost and faster time to market. The prudent manager can't afford to wait on the sidelines; he and his company need to get involved now. We suggest that you:

1. Educate your company about object-oriented technology
2. Identify where how your company can benefit from object-orientation
3. Invest only in products supporting open standards such Unix, X-windows, C++
4. Look for a database that implements a full object-oriented data model that can be implemented across multiple languages
5. Make sure your vendor is positioned to support standards for OO-DBMS DMLs and DDLs that will evolve over time
6. Look for an OO-DBMS that allows you to leverage your existing investment in hardware and software

---

Tom Atwood, *founder and chairman of Object Design, has pioneered the commercial development of object-oriented technology. Previously, he was one of the founders of Ontologic, one of the first companies to introduce a commercial object-oriented DBMS. He has worked extensively in the field of database development at Data General and Digital Equipment Corporation.*

Passing Messages

Passing Messages is a column occasionally published in HOOT that brings bits and pieces of information of interest to HOOT readers. If you have a suitable message, pass it on to the HOOT editor.

The first Smalltalk industry newsletter, HOOPLA!, will no longer be published by the Object-Oriented Programming for Smalltalk Application Developers Association (OOPSTAD) as OOPSTAD will cease operations at the end of the year. Citing the low level of membership, Carl Nelson thanked the pioneers who helped get it going. The activities of OOPSTAD are being carried on by user groups all over the nation and bulletin board services by vendors and by many of the electronic service companies (BIX, Compuserve, etc.)

Paul McCullough, one of the founders of Instantiations, Inc., has left Instantiations and joined West Coast Software, which is turning into a hotbed of OO activity.

Graphael, Inc., one of the early object-oriented data base suppliers, has closed its doors. As we understand it, a new company, Object Databases, Inc., is being formed to support the existing customers. The product itself is produced by a French company that is going strong. Furthur information can be obtained by calling (617) 354-4220.

The following is courtesy of Esther Dyson of Release 1.0 (22 Sept 1989):

Juan: What happens to spaghetti code when you go object-oriented? Does that fix everything?

Alice (sigh): No, you just end up with ravioli code.

Digitalk's Smalltalk/V PM has been nominated for a major award at the annual PC Magazine banquet at COMDEX. We won't know the result before press time but it's great to see OO products starting to receive national attention.

Speaking of Digitalk, George Bosworth (a founder and vary active at Digitalk) was quoted last month in an article on OOP and was listed as a MacApp Product Manager for Apple Computer. Are you moonlighting on the side, George?

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes,*** I want to be plugged into the latest thinking, and developments in the object-oriented technology. Immediately enter me as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	2 years (24 issues)
☐ ~~299~~ $249	☐ ~~$598~~ $498

outside the U.S.—add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT—U.S.currency drawn on U.S. bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card #_____ Exp. date_____
Signature _____

Name _____
Company _____
Address _____
City _____ State_____ Zip_____
Phone_____

For immediate service—call the Hotline 212-972-7055 or Fax 212-370-4983

HOTLINE on
Object-Oriented Technology
The International Newsletter

310 Madison Avenue
Suite 503
New York, NY 10017

BULK RATE
U.S. POSTAGE
PAID
PERMIT No. 47
MENASHA, WI

HOTLINE on Object-Oriented Technology — The International Newsletter

January, 1990 **Volume 1, Number 3**

Adopting Object-Oriented Techniques: Management and Organizational Impacts

by Marie A. Lenzi

Many attempts to implement object-oriented technology in commercial and engineering environments follow the same pattern as other technologies that changed the manner in which systems developers think and work. Usually, organizations design, implement and manage application systems projects using object-oriented programming systems (OOPS) without changing the development process itself. This approach results in several years of unnecessary expense, frustration, and unsuccessful projects.

The organization implementing object-oriented technology must overcome two different kinds of challenges: technical and managerial. As with structured methods and CASE, the technical issues involve learning the new technology, and then developing proficiency in using the technology. Success with the technical challenges alone does not result in implementation of a new technology. Object-oriented technology requires a different approach to thinking about the problem-solving and development process in relation to the new technology. Managers as well as developers, must experience a change in mind-set . The managerial issues are significantly more complex than the technical issues, and the managerial issues determine the success or failure of the use of the technology. These managerial issues, however, are often ignored altogether.

When first presented with the challenge of implementing OOPS, we elected to reuse previous experience in implementing new technologies. Our experience with implemented structured methods, in a wide variety of commercial and engineering environments, taught us that several crucial issues must be addressed:

1. Planning and budgeting
2. Resourcing
3. Estimating development projects

continued on page 3

Contents

Feature:
Adopting Object-Oriented Techniques: Management and Organizational Impacts
1

Thoughts From the Editor
2

Databases
The Object Oriented Database System Manifesto: A Consensus From Academia
6

Languages and Environments
Which Language to Choose, Revisited
9

Languages and Environments
On Language Wars
10

Publicity Corner
11

Conference Review
C++ at Work '89
12

Product News
13

Book Review
Object-Oriented Program Design with Examples in C++
14

Company Showcase
15

It has been hard for me to imagine that this is the third issue of the Hotline. It is moving more rapidly than I thought possible. While I have had full faith that the technology would continue to grow as rapidly as it has, my confidence in being able to bring it to you has not always been as strong. I have been pretty pleased with the results to date.

Some things do not works out as planned, however. Last month I promised you some articles in this month's issue that are not here. Now I know, and that will not happen again! Those promised articles are coming; I just won't say when!

This doesn't mean that we have nothing to present to you in this issue. This issue starts with a interesting article on the practical and realistic introduction of object-oriented technology into your organization by Marie A. Lenzi. Tom Atwood continues his series on object-oriented date bases with a discussion of the recent Object-Oriented Database System Manifesto.

A report on the the C++ At Work '89 conference is included and book reviews of two recent titles on C++. Our *Company Showcase* continues to present new companies and more new products are presented in *Product News*. *Publicity Corner* continues to review the industry literature for object-oriented references.

In our continuing effort to bring controversies to your attention, we have published a plea from Bjarne Stroustrup for an end to the language wars. I must add my voice to Bjarne's in cautioning users of the technology to find and use the most appropriate solutions available. It is very early in the life of this technology to establish broad standards. We have a long way to go in our understanding and in the development of tools and methodologies. Lots of alternatives still need to be explored.

A newsletter such as this Hotline is always looking for new material and new contributors. The Hotline will reflect the full breadth of activities of the technology only if folks from all the various camps and niches express their positions and contribute. I encourage you add your voice to the ones that have been presented. I am always looking for thoughtful articles on the issues as well as competent book reviewers and others wishing to report on object-oriented activities.

This issue marks the beginning of a new decade. We can all make the growth of object-oriented technology the largest story of the 1990's.

Jeff McKenna
Editor

HOTLINE on
Object-Oriented Technology
The International Newsletter

Jeff McKenna
EDITOR

ADVISORY BOARD
Jim Anderson
Digitalk, Inc.
Tom Atwood
Object Design, Inc.
Grady Booch
Rational
Larry Constantine
Consultant
Cort DeVoe
IBM
Adele Goldberg
ParcPlace Systems
Bertrand Meyer
Interactive Software Engineering
Reed Phillips
Knowledge Systems Corp.
Trygve Reenskaug
Taskon A/S
Bjarne Stroustrup
AT&T Labs
Dave Thomas
Object Technology International
Zack Urlocker
The Whitewater Group
Eugene Wang
Borland International

Richard P. Friedman
Group Publisher
Xavier Yaffar
Technical Advisor
Christine Coleman
Art Director
Melanie Younossi
Marketing Assistant

The Hotline on Object-Oriented Technology (ISSN#1044-4319) is published monthly by SIGS Publications Inc., 310 Madison Ave., Suite 503, New York, NY 10017. (212) 972-7055.
©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express persmission from the publisher.
Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30. Send address changes and subscription orders to HOOT, 310 Madison Avenue, Suite 503, NY, NY 10017
Submit editorial correspondance to Editor, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062

Regularly appearing in HOOT, this section introduces you to companies offering various products and services in the object-oriented field. Every company which sells an object-oriented product or service (old or new) will eventually appear in the Company Showcase.

Included here is a brief description of the company and its product or service, atributes, with name and phone listing for further contact. This section is provided as an information service to readers and no endorsements by HOOT are implied. Summaries are provided by the companies. Companies wishing to be included in the Company Showcase should send appropriate information to Jeff McKenna, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062. Tel. (503) 692-1889

Object International, Inc.

Object International, Inc., located in Austin, Texas, provides worldclass *OO Advisors™* consulting services—long-term, training, and advising on Object-Oriented development—one week per month.

The company recruits and retains top leading practitioners in Object-Oriented development. Each staff member has attained national or even international recognition for his/her expertise and pragmatic approach. Each staff member also fully dedicates his or her efforts to provide the very best for each client.

Leading the *OO Advisors* is Peter Coad, author (with Ed Yourdon) of "Object-Oriented Analysis" (Prentice Hall, 1990, available now).

For more information, contact:
Object International, Inc
8706 Silverhill Lane
Austin, TX 78759-7417
Phone & Fax (512) 343-6009

SCS Associates

SCS Associates, located in Boston's World Trade Center, is established to serve those organizations committed to the new generation of software development that is based on object-oriented design and related programming languages. SCS provides a research network for engineering decision-makers and project managers which enables a smooth transition to these technologies, including all the benefits of software reuse and gains in productivity that they bring.

The SCS research network offers:

•Contract consulting services on a project basis, staffed by professionals who have proven expertise in their fields.

•Retained and contingent search programs, including applicant screening, qualifying, and reference checking for permanent hire individuals.

SCS network spans both the national and international markets.

For more information, contact:
David Currie
SCS Associates, Suite 75
World Trade Center
Boston, MA 02210
Voice: (617) 439-5292
Fax: (617) 439-5455

HCR Corporation

HCR Corporation, founded more than twelve years ago, is recognized throughout the world as a leading software development company specializing in UNIX system software. Its services include custom porting of the UNIX operating system to various hardware platforms, developing optimizing compilers, software tools, and porting of relational database products to various UNIX systems.

Under the heading "advice and guidance," HCR provides consulting and training services to potential UNIX users and vendors who are planning to use the new technology, and who are grappling with the issue of where to start. Design of a UNIX-based network is often the answer. Network computing is another area in which HCR maintains state-of-the-art expertise. Additionally, HCR sells and supports a number of language and operating system products, including UX-Basic, HCR/PASCAL, dbXtra, HCR/C++, C Test Suite, and the 386/ix Product Family. The corporation is accustomed to licensing its proprietary technologies.

Since its founding in 1976, HCR has concentrated on developing a worldwide reputation for expertise in the UNIX marketplace. This reputation has led to its participation in UNIX standards committees in the USA and Europe. HCR's status as a standards setter is vital to its customers, who depend on HCR both for knowledge of UNIX internals and for insight into the future directions of UNIX.

HCR Corporation is pleased to have this opportunity to introduce itself and would welcome the opportunity to respond to any specific UNIX-related software requirements the readers may have.

For more information, contact:
HCR Corporation
130 Bloor Street West, 10th Floor
Toronto, Ontario
Canada M5S 1N5
Voice: (416) 922-1937
Fax: (416) 922-8397

Oasys

Oasys is one of the largest suppliers of comprehensive development tools for embedded systems and Object-Oriented Programming (OOP). They provide Green Hills native and cross compilers (C++, C, FORTRAN, Pascal), OASYS cross assemblers/linkers/librarians, simulators, a remote cross debugger, and Glockenspiel C++. Their tools support cross and embedded development for Motorola 680x0, 88000 and Intel 1860 and 80x86 target CPUs.

Oasys is the premier supplier of C++ software to the commercial market, supporting over 1000 customers implementing C++ applications since 1986. The Green Hills C++ compiler is the first to combine the optimizing techniques of high performance compilers with OOP. Oasys's clients work in industries such as aerospace, biomedical, defense, electronics, communications, transportation, banking, finance, telecommunications, and process control. Anyone involved with software development for CISC and RISC 32-bit microprocessors can benefit from Oasys's development tools.

Oasys compilers and tools are available on a variety of hardware platforms including DEC VAX, Sun, HP/Apollo, Apple, Unisys/Convergent, Motorola, and the IBM RT. Oasys also provides new development tools for their customers by porting existing products to new systems. The company is a founding member of 88open, Motorola's 88000 consortium, and is the exclusive supplier of five essential software tools for Motorola's new 88000 RISC chip.

For more information, contact:
Ann Bischoff
Director of Marketing
Oasys
230 Second Avenue
Waltham, MA 02154
voice: (617) 890-7889
fax: (617) 890-4644

International Meta Systems, Inc.

International Meta Systems (IMS) was founded in 1985, with the goal of developing and manufacturing a new generation of hardware and software designed to solve some of the major problems facing the maturing computer industry.

The forefront of its product line is their proprietary super microprocessor. Today's level of performance is the result of much refinement and tuning, and now serves as the platform for IMS's "computer on a board," the company's premier product, the MAX2.

The major impact of these IMS products is that many jobs which formerly required a mainframe for execution can now be performed on PCs or MAC/IIs with the IMS MAX2. The IMS product also distinguishes itself by providing features such as a tailored versions of Smalltalk and Prolog.

For more information, contact:
Paul D. Ray
Director of Sales
International Meta Systems, Inc.
23842 Hawthorne Boulevard
Torrance, CA 90505
voice: (213) 375-4700
fax: (213) 378-7643

Object Oriented Program Design with Examples in C++

(Addison-Wesley, 1989)

by Mark Mullin

Today we find many books on the market that explain the C++ language syntax and semantics. These books put much effort into explaining the details of the language, and there is often little room left in the books to explain how to use object-oriented constructs to one's advantage.

Mark's book leaves the details of the C++ language to other books, and addresses the more difficult question of how to construct an object-oriented program. This is a difficult topic to teach, compared with a new programing language. Learning a new language involves simply learning to express the solution to a problem in a new way. But teaching a new design approach involves teaching a new way to think about and to solve problems. Nevertheless, to use effectively C++, one needs to master a new thinking process.

As a teaching vehicle, Mark has selected a nontrival problem, which is vaguely stated in the first chapter (this is real life). The problem is refined and developed throughout the book, ending in code segments for the Zortech C++ compiler. With this example, abstract points are made concrete.

The information in this book is sound. One is exposed to most of the issues of constructing an object-oriented program. For example, what are the objects? What classes do they belong to? How do these objects fit into a class hierarchy? What member functions must be supplied by the objects, etc.? The inevitable discovery process for developing a preliminary system of objects is refined, as more is discovered about the problem.

The material in this book addresses object oriented design at the *-in-the-small* level and leaves the *-in-the-large* issues, and problem analysis issues, for other texts.

Two sentences from the book struck me as being quite significant:

> "A good object oriented design should not counter what you already feel is good programming practice, but is an extension of what you already do."

and

> "Put off writing the code until the last moment."

The first sentence is good advice for those who are building their skills in the object-oriented area. In some cases, one might become fascinated with objects and construct systems that are more complex than necessary.

The second sentence suggests that one can, and should, spend time thinking about the system of objects to be built before beginning the coding process. I agree with this advice, but this issue is hotly debated by many experts who have had good experiences evolving a program into existence. This, however, is a topic to be discussed elsewhere.

As was mentioned earlier, teaching design is difficult. Obviously, learning a new design approach is also difficult. This text requires study and review. It is not something that one can read in a couple of evenings. Read it and discuss it with a friend, or put a study group together; very few individuals are self-motivated enough to make it though this topic on their own.

This book assumes that the reader knows C and I would suggest that one read a C++ text and try writing a few small C++ programs before starting to read this book. The material discussed in the book will have a great deal more meaning if one has a little experience in his or her background.

Reviewed by
Norman L. Kerth, *Elite Systems*

Calendar

The HOOT calendar presents conferences and meetings that are focused exclusively on object-oriented technology. It should be noted that many other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to the Editor, HOOT, 8825 SW. Umatilla St., Tualatin, OR 97062

JANUARY 22-24, 1990

SCOOP-West—Techmart Center, Santa Clara, California. Co-sponsored by JOOP and the Wang Institute of Boston University. Tutorials, Panels, Product Presentations, and Exhibits. Keynote speaker—Borland's Philippe Kahn. Contact: Bob Daniels, Wang Institute of Boston University, 72 Tyng Road, Tyngsboro, MA 01879, (508) 649-9731.

MARCH 13-15, 1990

The Object-Oriented Symposium—Presented by Digital Consulting, Inc. and chaired by Larry L. Constantine, in Washington, DC. Repeat of the symposium presented in Dec '89. Contact: DCI, 6 Windsor St., Andover, MA 01810, (508) 470-3880.

FEBRUARY 5-9, 1990

MacApp Conference San Diego, Winter '90—Tutorials, workshops, general sessions to be held at Doubletree Hotel in San Diego, CA. Contact: MacApp Developers Assoc., 4327 Rucker Ave., P.O. Box 23, Everett, WA 98206, (206) 252-6946.

FEBRUARY 6-9, 1990

Software Development '90—Oakland, Convention Center. Annual 3½ day educational conference and product exhibition. Keynote—Microsoft's Bill Gates. Contact: Bob Saunders, Software Development Series, 500 Howard St., San Francisco, CA 94105, (415) 995-2471.

APRIL 9-11, 1990

2nd USENIX Conference on C++—San Francisco, CA. Tutorials, technical sessions. Conference chair—Jim Waldo, Apollo Computer. Contact: USENIX Conference Office, 22672 Lambert St, Suite 613, El Toro, CA 92630, (714) 588-8649

JUNE 6-8, 1990

Eurographics Workshop on Object-Oriented Graphics—Konigswinter, Federal Republic of Germany. Workshops cover standardization and OO languages for graphics. Contact: Marja Hegt, O-O Graphics Workshop, CWI, Kruislaan 413, 1098 SJ. Amsterdam, The Netherlands. Tel. 31-20-592-4058.

continued from page 1

4. Altering the project work-breakdown structure,
5. Overcoming the learning curve,
6. Managing and planning for reusability,
7. Managing the shifts in user expectations.

Planning & Budgeting

With the successful implementation of OOPS comes the opportunity to build more systems in less development time, thus a lower life-cycle cost. This successful implementation depends upon planning and budgeting for the cost of "tooling up" to use the new technology. Failure to account for start-up costs usually results in the underfunding of the basics required to overcome the technical challenges of implementation. Organizations planning to implement OOPS without spending money are not being realistic.

When first implementing OOPS, funding requirements for training, methodology changes, software tools, developer resources, external support expertise, and possibly, hardware, must be budgeted. With time, these costs are absorbed into ongoing procurement and support budgets.

OOPS effectively increases an organiztion's development capacity by allowing more systems to be built within current budget levels. However, total systems expenditures do not decrease if the priorities of an organization rate the building of more systems as more valuable than budget cutting. Because of this, greater emphasis must be placed on planning and budgeting for the long-term based on the needs and expectations of the organization.

Organizations taking advantage of the increased development capacity offered by the mastery of OOPS, will probably not see a reduction in total maintenance expenditures. The maintenance per system should decrease, but, because more systems will be built, there will be more systems to maintain. The overall maintenance budget probably will not decrease.

Resourcing

Resourcing (staffing) is currently the most difficult issue to be addressed. The technology is not only new, it is different. Because of this difference, few real experts are available for hire. Even more unfortunate, many who sell themselves as experts have never delivered products to market using object-oriented technology.

Although OOPS technology is more approachable for people with backgrounds in C, Pascal, or Modula programming, experience with other languages is not an impediment to learning the new development tools. More critical than language background is the selection of people with flexible minds. People able to change and adapt, who can view "mistakes" as vital to the learning process, are best suited for making the change from procedural to object-oriented development.

Knowledge of, and experience with, structured methods significantly eases the transition to OOPS. Current object-oriented analysis and design techniques are often based on existing, proven, structured methods. OOPS is, in fact, the realization of such basic concepts as encapsulation and information hiding.

Estimating Development Projects

Because OOPS development differs substantially from procedural language development, new project estimating techniques are required.

Project estimating is a function of three factors:

1. The organization's political and technical behavior
2. The organization's experience with effective estimating practices
3. The organization's experience with the technology in use.

This is a good news/bad news situation. The bad news is that OOPS probably can do nothing about corporate politics. The good news is that OOPS does provide an opportunity to improve estimating skill, once familiarity with the technology is developed.

OOPS requires estimating changes throughout the development cycle. With structured development techniques, we experienced a "front end" loading of the development life cycle—effort shifted from coding and testing to analysis and design. With OOPS, experience to date suggests that effort will be even more focused upon what is traditionally considered to be the design stage.

Part of the increased effort during design comes from prototyping. In many OOPS environments, prototyping provides a cost-effective tool for the better understanding of application requirements and design. This is particularly true in pure (as opposed to hybrid) environments. The payback comes in programming, and later in enhancing the finished product. Accurate estimating of OOPS projects requires experience with prototyping.

Good OOPS design results in many small, loosely-coupled pieces of work. Experience teaches that many smaller components are easier to understand than a few complex monoliths. The estimator must suppress the temptation to recalibrate for use with OOPS the traditional estimating techniques which are based on counting program modules, assigning complexity weights, and multiplying by units of average time per module needs. The historical (experiential) database of project estimates versus the actuals one may have been building is not directly transferable. A new database must be established for post-learning-curve OOPS projects.

Increased modularity also increases team interaction. Individuals work on small components, then join together for integration, testing and debugging. The cost of increased communications is offset by greater clarity of focus for the project and the product. When developers understand what is being built, and the components are small enough to be built correctly, the actual time cost of integration and debugging decreases. As an added plus, end product quality increases.

Another consideration during initial attempts to estimate OOPS development is the non-traditional learning curve. In traditional language learning, effort is focused on understanding the syntax. Experienced OOPS developers consistently report that, in general, learning the new language syntax requires far less time than discovering and understanding the contents in the class library. Discovering which pieces already exist for reuse is the most significant challenge for developers first encountering OOPS.

Experience with several projects will help both estimators and developers become more comfortable with this part of the process. You will become able to assess what is required and achievable with OOPS in your organization. Learning to estimate OOPS projects is part of the learning curve.

Changing the Work Breakdown Structure

OOPS implementation requires changes to existing systems development methodology. The degree of change depends upon both the methodology and the OOPS environment used.

Adaptability is crucial. As with procedural development, no "right" method exists. Several OOPS development methodologies are currently under development; all are rooted in known and proven structured methods; none have been rigorously tested under real application development conditions.

One clear change with OOPS is a need for greater interaction between programmers and designers. Since the physical implementation of OOPS applications can more closely model the real world, implementation issues and alternatives cannot be ignored until the project is "handed over" to programmers. This increased interaction has several benefits: involvement breeds commitment; programmer understanding of the application increases with greater upfront involvement; true prototyping can be undertaken effectively with the teaming of both programmer and designer skills.

Although the modeling requirements for OOPS have not been solidified as yet, substantial agreement among practitioners does exist on certain basic, familiar modeling principles.

Requirements still must be fully and precisely specified. Existing techniques for modeling the requirements apply because we are still solving the same kinds of problems. Because object-oriented technology models the real world more closely than functional approaches, it becomes natural to assess requirements with users in terms of "objects." The rigor of structured method is not diminished by talking "objects" with the user.

Planning time for good, thorough design promotes reusability and maintainability. Managers must reserve time and effort during the design phase for imperative "new" activities.

More attention must be paid to system architecture. In object-oriented technology, architecture is tightly linked to the class hierarchy. Developers must determine which new classes, or changes to existing classes, will be required, and how these new classes will interact with existing classes. Because of message passing and inheritance, placement of new classes within the hierarchy is a far more significant task than the mechanics of installing a subroutine in a library.

OOPS design is more akin to long range systems planning than traditional module design. Development of reusable code and classes is predicated on understanding the "big picture." The current and future class structure of an application must be considered at all times. For this reason, development of an object-oriented application in a vacuum is potentially more costly than development of a similar project in a procedural language. An isolated (single application or person) perspective on the class structure precludes systems planning foresight, reusability, and alternative model structures. Isolation commonly causes design problems such as class redundancy and inconsistent message passing protocols.

Reusable code is not free. In addition to the time required to design reusable components, we have found that reusable classes go through several iterations in the process of becoming reusable. The designer cannot always anticipate that a component will be needed elsewhere. For such nonobvious shared components, tasks for redesign and rework of existing classes must be figured in to the work breakdown process.

Prototyping tasks are added to the work breakdown structure. Prototyping does not substitute for design or programming. Programming tasks cannot be dropped on the assumption of "getting some use" from prototype components. Misuse of prototyping perpetuates the illusion that the "finished" product will arise from the ashes of exploration. On the contrary, prototyping in lieu of design results in cumbersome class trees, incorrect inheritance, redundancy, and overly-complex applications. Thorough design is backed up by prototyping to test and verify both the concept and the implementation results.

Coding and testing both will experience an increase in the number of discrete tasks. This would be expected in a development environment, which promotes applications constructed of many small modules. In this environment each module must be designed, built, and tested. Interfaces among components, too, must be tested. Leverage is gained from content, not count. Each component is simple and of clear purpose when compared to traditional monoliths. Clarity promotes understanding; understanding promotes working products.

The best returns from OOPS developer resources are gained from coherent and meaningful work assignments. Task allocation should be along class tree boundaries. Developers are more effective and productive when they are working on clusters of related components. This approach facilitates small module coding and testing as well.

Product Announcements should be sent to the Editor, HOOT, 8825 SW Umatilla St., Tualatin, OR 97062.

IMS Adds 70 MIPS Processor to MAX2 Family

International Meta Systems, Inc. (IMS) announced the newest member of the MAX2 family of object-oriented development platforms. The MAX2 contains a VLSI microprocessor, running at 35 MegaHertz with 2 instructions per clock cycle. The Writeable Instruction Set Computer is programmable at the instruction level, allowing programmers to change the instruction set for specialized application tuning.

It is available on a single board computer in two versions: one for the IBM PC/AT and one for the Apple Mac/II. The boards include 4 to 64 MB of RAM, necessary host interface software, and a MAX2 instruction set, optimized for object-oriented software. Smalltalk/V, Prolog, and LISP are high level languages available on the MAX2. An assembler and source level debugger are available for customizing the instruction set. Libraries are available for windows, mouse support, and bit-mapped graphics.

Shipment is 30 days ARO. The price was not available.

For more information, contact:
Paul D. Ray
Director of Sales
International Meta Systems, Inc.
23842 Hawthorne Boulevard
Torrance, CA 90505
voice: (213) 375-4700
fax: (213) 378-7643

Objective-C™ 4.0 for MS-DOS

The Stepstone Corporation announced that the Objective-C Compiler Version 4.0 is now available for MS-DOS and is compatible with MS-Windows. The Objective-C Compiler is an object-oriented superset of ANSI C which provides C language developers with the full power of object-oriented programming, including: objects, classes, inheritance, messaging, plus the flexibility to program in a dynamically bound environment. While providing object-oriented support for the developer, Objective-C Version 4.0 also recognizes requirements for efficient execution at runtime and integration with conventional applications. It provides support for static binding at the programmer's discretion and compatibility with C for ease of integration.

The MS-DOS version requires a PC/AT compatible platform with MS-DOS 3.x and Microsoft-C 5.0 or 5.1. MS-DOS based development machines for Objective-C require 2 Mb of RAM and 2Mb of hard disk. Executables generated by Objective-C for MS-DOS are compatible with the OS/286/386 DOS Extender Kernal available separately from Eclipse Computer Systems. Using the Eclipse Extender, an Objective-C application may address up to 16 Mb of Extended Memory. Objective-C applications smaller than 640K do not require the DOS extender.

The Compiler packaged complete with ICpak101 basic data structure library is priced at $249. Availability was not announced.

For more information, contact:
The Stepstone Corporation
275 Glen Road
Sandy Hook, CT 06482
voice: (203) 426-1875
fax: (203) 270-0106

Oregon C++ for VAX/VMS

Oregon Software, Inc., announced the availability of Oregon C++ for VAX/VMS. Oregon C++ is the first true commercial C++ compiler for VMS. Major features include a source-level debugger, the NIH OOPS class library, and support of shared libraries and VAX C calling sequence. Oregon C++ can call any DEC language as well as Oregon Software's C, Pascal-2 and Modula-2. Oregon C++ for VAX/VMS runs on VMS 5.0 and later. The Oregon C++ compiler also includes an ANSI C K&R C compiler at no additional charge. The general release in December will include complete compatibility with V2.0 of AT&T's cfront and a task library.

The VMS implementation joins the Oregon C++ Motorola 680X0 Delta Series, Sun-3, Sony NEWS, HP9000/300, NCR Tower 32 Series, and Xenix/386 compilers previously released. The portability of the Oregon C++ compiler allows companies that use a variety of computers to have the same application on different machines. Field testing began this month.

Oregon C++ is sold through Oregon Software and its international distributors. License fees in the U.S. range from $2,000 to $34,000, depending on machine, cluster, or network configuration.

To aid customers in developing proficiency in the C++ language, Oregon Software also offers public and in-house C++ training seminars in both the U.S. and Europe.

For more information, contact:
JoAnn Bertram
Oregon Software
6915 SW Macadam Ave., Suite 200
Portland, Oregon 97219-2397
Voice: (503) 245-2202
Fax: (503) 245-8449

C++ at Work '89

by Bruce Eckel

The recent *C++ at Work '89 conference,* held in Boston, Massachusetts, was pitched as a learning forum toward technical managers, rather than as a forum for new technology. This emphasis was for three reasons. First, the conference makers (SIGS Publications) did not want to compete with the venerable *Usenix C++ Conference,* which was the original forum for technical papers and has been held for the last three years.

The second reason for the emphasis of the Boston Conference was to give the technical manager an alternative to spending the money for a consultant to find out about C++ for him, and to help determine whether it is appropriate to start using C++ in his or her company. The cost of going to the conference for three days was significantly higher than most technical conferences, but it was far cheaper than bringing in a consultant for a few days, so it serves this purpose well.

The third reason for the emphasis was because of the plethora of C++ training courses that have been appearing of late. With the increasing need for C++ training, it is difficult for a manager to determine the quality of the training he will receive based upon an advertisement or a discussion with a salesperson on the phone. Most training companies charge a considerable fee, and which company to use is an important decision. *C++ at Work* provided a forum for trainers to "demonstrate their wares," so the technical manager can know what he or she is getting. Again, the conference was much less expensive than the cost of a training course.

I went the conference a skeptic, since I thought there would not be anything sufficiently technical to keep my interest. It was good to go to this conference as a consultant and author, , but, generally, I was prepared to be bored.

I was, however, pleasantly surprised. The quality of the conference attendees (as well as the presenters) was high, and this made for many interesting hallway conversations as I get most of my best information in conference hallways! Bjarne Stroustrup (the creator of C++) and Andy Koenig (who works with Stroustrup and contributes to the language) were both there, as well as Walter Bright (author of *Zortech C++*). These men provided fascinating conversations.

The bulk of the presentations were introductory. I popped into many and found them well-attended, with much of interaction among the presenters and the attendees. In addition, there were enough technical presentations to keep me quite satisfied.

Sam Druker and Steve Teale, technical staff-members at Zortech, gave an interesting presentation entitled "Black Holes in C++," which was about the common places people get stuck in the language. Mike Ball (owner of Taumetric, which creates C++ compilers) discussed "what goes on in your C++ compiler." In addition, technical papers were presented; George Beshers talked about "mobile objects in C++," Bjarne Stroustrup and Andy Koenig presented a paper on exception handling (a feature which will be included in a future version of C++), and Mike Vilot talked about object-oriented design.

The most entertaining and enlightening speech was given by Andy Koenig (A Second Course in C++). Andy is enjoyable to listen, partly because he's so relaxed with the language—he doesn't need to prove that he knows it, and partly because he's more concerned with not leaving anyone behind than in covering vast amounts of material ,or in presenting a really big or complicated example. This lead should be taken by other speakers; it is better to give the audience a few clear concepts that they can take away with them than to risk swamping them by covering too much material.

The show floor included most, if not all, the compiler vendors as well as tool makers and training companies.

The last item in the conference was a forum discussing the future of C++, which included Martin O'Riordan (formerly with Glockenspiel, now with Microsoft), Gene Wang (Borland), Bill Gibbons (Apple), Andy Koenig (AT&T) and others. This provided some spirited interaction, especially when O'Riordan said that Microsoft believed C++ compiler speed and optimization was "a holy grail" that didn't benefit the end users. Gene Wang responded that Borland would gladly take up the crusade (it may be, however, that Microsoft merely said this to cause Borland to relax and to think that they've already won). The appearance of these players on the forum indicates their commitment to C++. I also met others who are working on compilers; Metaware, for instance.

I was pleased to have attended this conference. *C++ at Work '89* fills a definite need in the critical process of C++ education.

Bruce Eckel *is the author of* Using C++ *(Osborne/McGraw-Hill 1989) and the is owner of Revolution2, a C++ consulting and training firm. He can be reached at (215) 444-0828, on BIX as beckel, on Compuserve as 72070,3256, on the Internet as 72070.3256@compuserve.com, or by mail at Revolution 2, P.O. Box 760, Kennett Square, PA 19348, USA.*

Overcoming the Learning Curve

Learning is essential in order to keep current with changing technology. Managers and technicians cannot stay current with an entire field of knowledge unassisted. With complex and rapidly changing technology, "try it and see what happens" costs more than its minimal returns. Formal (classroom) training is the cost-effective alternative to obsolescence, and is a valuable tool for developing a command of new technology.

Training courses must be professionally developed and delivered by presenters who are experienced both in both the theory and the actual use of OOPS. The entire project team will require appropriate training in the new technology; this includes training for managers, analysts, designers, testers, and support staff, as well as programmers.

Funding and resourcing is not adequate management support. Managers must develop sufficient understanding of the technology in order to propel the cultural changes needed to implement OOPS. Management must be able to work knowledgeably with the OOPS project team.

While formal language training facilitates migration from procedural to object-oriented languages, the need for OOPS design training is perhaps not so obvious. For those persons with strong backgrounds in structured methods, much basic knowledge is transferrable. The specifics of OOPS design technique, however, still must be learned. Cost-effective use of the technology, not to mention such prime paybacks as code reusability, will be achieved only through the development of sound, consistent design techniques.

Support may be needed beyond classroom training. Keep in mind that developers and managers alike must cope with substantively different concepts than those with which they have been familiar. When confronted with the frustration of new ways of thinking, people will return to "old" approaches. Support reduces the frustration, helps restore confidence, and provides technical answers and direction. When an organization is first implementing OOPS, this support must be received from outside expertise. As internal expertise develops, healthy organizations emphasize the use of "internal consultants" for this supporting role. While providing this expertise may appear costly at first glance, the real cost is the difference between success and failure in implementing OOPS.

Most learning occurs beyond the classroom. Assimilation and adaptation depends on real work experience — learning from exploration and mistakes. With the introduction of a new technology, the first few real projects will take longer because of this learning process. Failure to plan for learning time will not eliminate the need for learning and rework. Where learning time is not provided, and rework is not done when needed, the new technology usually is perceived as the culprit. Yet this is entirely a management problem. Managers need to schedule in learning time or they can expect to overrun the schedule.

Managing and Planning for Reusability

Because it takes several iterations before a class is truly reusable, mass-scale reusability does not occur with the first OOPS project. As the team gains experience, and the class library matures, developers increasingly find opportunities for reuse.

Short-term management expectations must be tempered. An eye on long-term planning for reuse is essential. Skills in designing for reuse must be trained. As with the development of other technical skills, both learning time and resources must be provided for by the manager who is seeking to take advantage of reusability.

Managing Shifts in User Expectations

Users in most organizations have become acclimated to late product delivery, unpredictable quality, and budget overruns. When these fhings first occours, the result may be "user shock." Experience with OOPS to date, however, suggests that most users are unprepared for good systems delivered on time!

Timely availability empowers an organization to respond to changing market conditions. Predictable and affordable cost places systems solutions within the grasp (and budget) of managers who would not otherwise be likely to risk commissioning development projects.

Managed correctly, OOPS provides the opportunity to create a new relationship between systems developers and users. User/developer tension around schedule and cost are symptoms of a more fundamental problem: limited and inadequate interaction between the subject-matter experts (users) and technology experts (developers). This tension results in poor product quality and poor fit to needs. At issue is the need for improvement in the development process itself.

OOPS is best suited to interactive design and development, prototyping, and building modular components. Leveraging this, the manager can organize projects to deliver meaningful components of the whole product. Multiyear, monolithic development can be avoided. Checkpoints can and *should* be built around prototype evaluation. This approach alone increases developer/user interaction, reduces the size of misunderstandings to repairable levels, and provides more opportunities to build the communications among users and developers.

Wrap Up

The greatest inhibition to successful implementation of team technology has been the lack of management drive.

are susceptible to failure upon the slightest indication of erratic management. If a more effective, successful, development process is the objective, management has no alternative but to embrace staunchly this commitment.

OOPS presents an opportunity for the implementing organization to improve the emotional tenor of its systems development process. On-time delivery of small, loosely-coupled, and cohesive systems, that function as expected, will substantially enhance the image of systems developers. Such improvement will be experienced only by those committed to change. Attention to the technology is critical; conscious management of the implementation process will determine the final outcome.

Our industry has learned painful lessons in attempting to implement structured methods and CASE. The changes required for OOPS are very similar in an organization must cooperatively alter the way it works. If the implementation is consciously managed, much of the chaos, discord, and squandered expense, experienced when implementing structured methods and CASE, can be avoided.

Marie A. Lenzi *is a principle of Syrinx Corporation, who are specialists in object-oriented technology, development, training, and consulting. She can be reached at (212) 606-2019, or by writing to 210 East 86th Street, New York, NY 10028, USA.*

Data Bases

The Object-Oriented Database System Manifesto: A Consensus From Academia

By Tom Atwood

As object-oriented data base management systems (OODBMS) move from the laboratory into the world of commercial applications, engineering managers are confronted with the need to distinguish among the dizzying array of database products that call themselves "object-oriented." Help is at hand, in the form of *The Object-Oriented Database System Manifesto* published by a distinguished group of OODBMS researchers, including:

Malcolm Atkinson of the University of Glasgow;
Francois Banchilon of Gip-Altair;
David DeWitt of the University of Wisconsin;
Klaus Dittrich of the University of Zurich;
David Maier of Oregon Graduate Center; and
Stanley Zdonik of Brown University.

In this column, I will summarize the *Manifesto* and comment on its implications for managers. Next month, with the help of Richard Soley of the Object Management Group (OMG), I will present OMG's plans for defining OODBMSs. OMG is an industry group dedicated to object-oriented technology.

The following month, I will compare and contrast the Manifesto with the OMG report, and will examine the implications for the future of OODBMS standards and technology.

The *Manifesto's* Golden Rules

The *Manifesto* is organized according to a series of characteristics of object-oriented databases. The first set of thirteen Golden Rules are mandatory for any OODBMS satisfying the *Manifesto*. The Golden Rules are divided into two groups: eight rules make a system "object-oriented," and five make a system a DBMS. Subsequent to the description the Golden Rules, this article will describe five optional "goodies" that can significantly enhance the power and functionality of an OODBMS.

Before summarizing the Golden Rules, I will first describe what they are not. The rules are not the equivalent of Codds Rules for relational DBMS, or a standard similar to SQL. Instead of postulating a highly rigorous set of rules, the *Manifesto* proposes a broad set of requirements that a DBMS can satisfy in order to be considered an OODBMS. This approach is necessary because OODBMSs are inherently different from relational DBMSs. More important, they incorporate general programming facilities, and hence can be used to build arbitrarily complex computations. In contrast, relational systems are not powerful enough to represent arbitrary computations. Moreover, the distinction between the language and the DBMSs is vastly different in an object-oriented paradigm. Consequently the Manifesto must offer guidelines that cover the combination of language and DBMS, which blurs the distinction between the two.

Publicity Corner

We are continuing to receive our regular mail from the clipping service, and this month we received about 110 articles. The distribution was as follows: approximately 50% of the announcements were for various OO based products; 10% were marketing hype and related announcements; about 30%were generic information.

A few observations were gleaned from this material.

The larger computer companies continue to support object-oriented technology. This month, DEC continues its announcement of DECdesign Version 1, an object-oriented software development environment with a graphical user interface (*Electronic News,* Oct 16). DEC also discussed its ongoing work with Atherton Technology, which provided a software backplane supported by an object-oriented database *(EE Times,* Oct 16).

Unisys has announced that their Linc product will move toward objects, with the Linc Design Assistant using an object-oriented language *(Computerworld,* Oct 20).

4GL providers have also joined the bandwagon by announcing the movement of their products toward objects. Information Builders, Inc. has announced a three phase plan to move Focus, their 4GL, toward full object-orientation by 1991.

An interesting look at the efforts of larger companies was presented in an analysis by Joshua Greenbaum in the Oct 23 issue of *Computer System News.* In this article he analyzed the job opportunities presented at the recent OOPSLA conference. "There are lots of job openings with some large and important companies doing some exciting work." He found that Rockwell is looking for expertise in OODBs; KPMG Peat Marwick is looking for object-oriented Mac programmers; Apple and US West are looking to use objects in telephony; Texas Instruments, Inc., wants it all; OOP, CASE, OODB, CAD, and CAM!

The Diebold Group's latest study of current software trends and their implications lists two emerging development methods as significant: Rapid Prototyping, and Object-Oriented Languages. They also list Prototyping tools and Design Description tools as the most important tools now emerging. They see Object Prototype Libraries and Generic Application Kits as important products anticipated in the future. All this was reported in *Computer Age - Software Industry Report,* Nov 6.

We saw the first CASE vendors at OOPSLA this year, and the literature has reflected the growing interest in how CASE and OOT are joining together.

An in-depth interview with CASE Research Corporation's Greg Boone and Vaughan Merlyn in the October issue of *News 3X/400* mentions object-oriented technology as complementary to CASE. Boone and Merlyn see difficulties in absorbing the new technology. "...traditionally trained programmers and analysts sometimes have trouble absorbing the newer methods... The paradigm shift from structured programming to object-oriented programming is even more challenging (than the shift from procedural to nonprocedural programming)." They point out that the major obstacles are not technical, but managerial.

The November issue of *Expert System Strategies* contains a lengthy article titled *OOPS, CASE, and Expert Systems,* that continues to explore this topic. The author gives a full review of OOPSLA, and connects the technology and products that he observed to both AI developments and CASE. Coming from this point of view he sees major weaknesses in the state of the art in OOT. "First, there is no acceptable methodology for identifying objects ... Second, most object oriented programmers don't know what inferencing is or how it could help them." He faults the AI and the CASE folks, as well as the OO folks, for their inability clearly to identify objects. He also laments the lack of inferencing as a powerful aid in the prototyping process. All in all, it is an interesting article that brings together the three major advances in software development technology's recent history.

Plug into the

HOTLINE

for the latest trends, issues and news.

Call
212-972-7055

For information about reprints, please contact

Professional Press, Reprint Resources
155 Commerce Dr.
Fort Washington, PA 19034

(215) 643-9143
Fax (215) 643-9164

On Language Wars

By Bjarne Stroustrup

There seems to be a misconception that C++ is supposed to be Smalltalk or that Smalltalk and C++ are in some way supposed to be the same. One might summarize the misconception in a "sillygism:"

C++ is Object-Oriented
Smalltalk is Object-Oriented

therefore:C++ is Smalltalk

It follows from this view that one should be able to work in exactly the same way with implementations of the two languages and that one should be able to accomplish the same tasks with either language. This view implies that when one cannot use the languages interchangably, someone has goofed. I disagree with this premise. People who naively approach C++ and Smalltalk with this premise are going to be disappointed.

C++ was not designed to be an imitation Smalltalk. Had I tried that, I would naturally have produced something much closer to Smalltalk. I do not see a need for imitation Smalltalks. Smalltalk is still the best Smalltalk around; it is a very nice language/system and its evolution is in capable hands. The same has been said about C++.

So,
C++ is not a good Smalltalk; it was not meant to be.
Smalltalk is not a good C++; it was not meant to be.

There are two fundamental differences between C++ and Smalltalk: (1) C++ is a language; Smalltalk is a system. (2) C++ is statically typed; Smalltalk is dynamically typed. These first order differences are derived in part from needs and application areas, and in part because they represent alternative strategies for attacking problems. Many differences follow from these differences. For example, in general:

- it is harder to built a high-quality ("modern, Smalltalk-like") programming environment for a statically typed language;
- it is harder to provide a high-quality programming environment for a language used on a diverse range of systems (rather than being a system in itself);
- it is easier to design "standard libraries" in a dynamically typed language;
- it is harder to efficiently produce running code for a dynamically typed language;
- it is harder to cooperate with "traditional" systems from a dynamically typed language embedded in a non-traditional standard environment;
- it is easier to manage the integration of large systems written by many programmers if statically typed interfaces are used.

The list could be made much longer, but as it becomes more specific, the specifics become difficult to understand in isolation. Also, as the list becomes more specific, data is harder to come by, and the issues become more sensitive to matters of system maturity, application area, economics, and the background of the users.

It is clear that both classes of languages/systems evolve to compensate for their weaknesses. It is also clear that the average proponent of one language/system knows his/her own language much better than the alternatives, so that much criticism is based on ignorance about the strengths of the alternatives, and is often leveled against obsolete versions of the language/system.

The style of programming and the approach to design is (and I think ought to be) strongly influenced by the language/system used. In particular, C++ design typically requires a larger "up front" design effort than does Smalltalk design, and the specification of strong statically typed interfaces is usually the key element of C++ programming. Ignoring fundamental differences in one's basic tools leads to poor design, and to inefficient and poorly structured code.

The choice of language/system for a job should be done with a degree of understanding of both the strengths and the weaknesses of the alternative languages/systems. In the few cases where genuine data can be obtained, it should be taken seriously, and not merely dismissed in favor of philosophical arguments.

There is no reason to believe that a single language/system will be the best choice for all applications—even if a particular user may have the luxury of working exclusively on projects for which a single language/system has a fairly constant overall advantage.

A novice's questions deserve a better answer than a polemic blast. The various "object-oriented" languages do have something basic in common (for example, see my paper *What is Object-Oriented Programming?* in *IEEE Software Magazine*, May 1988), and if people approaching OOP are welcomed by a blast from a parochial religious war, it reflects badly on all languages.

There is no language/system in existence that can serve the range of applications/needs that is currently being served by various "OOP languages." C++ and Smalltalk were *deliberately* placed at the extremes of the spectrum of reasonable design choices for practical languages at the time they were both developed. This makes comparisons between them interesting, but tricky. They support each other's growth as much as they compete for resources. My conjecture is that there are many more Smalltalk programmers now than there would have been, had C++ not existed—and vise versa.

The Object-Oriented Golden Rules

Rule 1: Thou shalt support complex objects

This rule states that it shall be possible to build up complex objects from other objects and primitive types via "object constructors," or "type generators." It must be possible to represent tuples (or records), sets, bags, lists, and arrays as object constructors. Moreover, it must be possible to store any object in any of these type generators. Language constructs such as C structs and Pascal records satisfy this requirement. Interestingly, the relational model does not satisfy this requirement, because the only sets allowed are sets of tuples. In addition to mandating such complex objects, the rule requires that it be possible to define operators that traverse complex objects.

Rule 2: Thou shalt support object identity

This rule states that complex objects shall have an identity independent of the values they contain, which allows objects to be shared. Typically, programming languages embody the notion of object identity in *pointers.* Object identity gives rise to two levels of object equality: objects may be identical, or they may contain the same fields. The first level is computed by comparing pointers to the two objects, while the second level requires comparison of the fields of the two objects.

Rule 3: Thou shalt encapsulate thy objects

One of the familiar principles of object-oriented technology is the separation of the object interface and the object implementation. The object interface consists of a set of functions providing a formal definition of the contract to be satisfied by the object. The object implementation consists of functions and states that are completely invisible to the clients of the object. This notion of encapsulation allows the object implementation to change in radical ways, without changing the way in which any application uses the object. Languages like C++, which provide for a distinction between public and private members, provide encapsulation. The authors note that there are applications, such as database browser tools, which must violate the encapsulation rule.

Rule 4: Thou shalt support types or classes

Whereas most of the rules constrain the make-up of an OODBMS system, this rule extends the notion beyond most familiar language examples. The rule says that, as long as encapsulation is supported, a language may either support declaration of data types that statically describe the make-up of an object, or support the more dynamic notion of a *class,* in which the object creation paradigm is to make copies of existing objects. One should not be confused by the word "class": C++ classes are types in the terminology of the paper.

Rule 5: Thy classes or types shall inherit from their ancestors

Traditionally, object-oriented systems have satisfied two axioms: encapsulation and inheritance. This rule introduces the requirement for inheritance. As the authors state, there are several aspects of inheritance. *Substitution inheritance* requires that any operation on an ancestor is supported on a descendant. *Inclusion inheritance* requires that any descendant type be a subtype of all of its ancestors. *Constraint inheritance* provides for definition of subsets of a type, such that every element of the subset satisfies some constraint. *Specialization inheritance* occurs when a subtype is created by "specializing," or adding information to, elements of the ancestor type. The authors do not choose among these inheritance models; any of these models can satisfy the rule, and essentially all object-oriented systems provide one or more of these mechanisms.

Rule 6: Thou shalt not bind prematurely

This rule mandates the dispatching of operations on objects, so that applications may manipulate a variety of object types, without having explicitly to name the operations on each type. This is a major tenet of object-oriented systems. Operation dispatching is described as "late binding" because the operation actually invoked is determined not at link-time but at run-time. In the absence of late binding, applications are littered with "case statements," which pair object types with operations, and must be updated whenever a new object type is defined

Rule 7: Thou shalt be computationally complete

This rule requires that any computable function must be expressible in the language. The rule is satisfied by essentially all programming languages, except for SQL, which is the popular relational query language.

Rule 8: Thou shalt be extensible

This rule dictates that there be no distinction in usage between system-defined and user-defined types. Rather, all distinctions between them are masked by the implementation, so that they look the same from the point-of-view of the application.

The DBMS Golden Rules

The golden rules that specify DBMS capabilities are those that are familiar from conventional database technology:

OODBMSs must store data persistently; they must provide indexing and clustering services that speed access to large databases; they must provide simultaneous access to multiple users; they must survive hardware and software failure; they must provide an interface for database queries.

Rule 9: Thou shalt remember thy data

OODBMS must provide for *persistence of data,* in the sense that persistent data continues to exist after the application creating the data has terminated. Moreover, *persistence must be orthogonal to type* so that any object may be stored persistently without special declarations or object translation. Finally, persistence must be *implicit,* ruling out explicit application commands to move or copy data in order to make the data persistent.

Rule 10: Thou shalt manage very large databases

Operations such as index management, data clustering, data buffering, access path selection, and query optimization are essential components of high-performance OODBMSs which are supporting large databases. This rule dictates that such operations be performed by the OODBMS, such that they are invisible to the application. There must be a clear separation between the logical and the physical level of the OODBMS.

Rule 11: Thou shalt accept concurrent users

One of the main virtues of DBMSs is that they allow multiple users to access the databases simultaneously, in such a way that they don't interfere with each other. The phrase "serializable access" is the database jargon for this kind of simultaneous access.

Rule 12: Thou shalt recover from hardware and software failures

This requirement is also a familiar one to those who have used traditional database systems. It must be possible to recreate any database in some coherent state, following a software or hardware, processor or disk failure. By "coherent state," we mean that the contents of the database after recovery reflect some single instant of time in the history of the database before the failure.

Rule 13: Thou shalt have a simple way of querying data

It must be possible to state database queries in a high-level and concise form, which is independent of application, and is susceptible to query optimization. The authors allow either programmatic interfaces or graphical interfaces to provide the query facility; the main point is that it be fast and easy to formulate and execute *ad hoc* queries efficiently.

Optional Features: "The Goodies"

The authors are aware that the Golden Rules will not provide an easy choice between OODBMS capabilities, because the Golden Rules are likely to be supported in some fashion by every offering. In contrast, the five optional features will provide more of a basis for comparing OODBMSs.

1: Multiple inheritance

The authors state that there is no firm consensus in the object-oriented community with regard to multiple inheritance between object types. Hence they regard support for multiple inheritance as being optional. We note that for systems that support multiple inheritance, such as C++ Release 2.0, it is possible to write applications using either single inheritance or multiple inheritance, while this is not possible using systems supporting only single inheritance.

2: Type checking and type inferencing

Although the authors classify compile-time type checking as an optional feature, they explicitly state that if more compile-time checking is provided, it is better. *Type inferencing* refers to the ability of the language compiler to deduce given declarations of base types. Sets of base-type objects represent conceptual derived types, based on how they are used. Type inferencing is very much a research topic, but the authors again say, the more the better.

3: Distribution

Any OODBMS may be *distributed,* in the sense that it provides simultaneous access to users running on different machines. There are many kinds of distribution, and some kinds of distribution can be viewed as a consequence of the concurrency requirement of Golden Rule 11. Nonetheless, the authors regard distribution as an optional feature.

4: Design transactions

For many OODBMS applications, the classical transaction model used for business DBMSs is not sufficient. These applications need some form of nonserializable "design transactions."

5: Versions

Hand-in-hand with design transactions is the ability to store efficiently and access various revisions of a single design. The authors applaud the utility of such a versioning capability, while noting that the capability is not in the set of core requirements.

Comparison with Relational Models

How do the rules for OODBMSs compare with the rules adopted for relational systems? The most significant difference is that the Golden Rules above support more complex and richer types and operations on types than relational systems. OODBMS allow:

- arbitrary object constructors, as opposed to the tuples and sets-of-tuples facilities of the relational type system;
- the definition of both state and operations, while the relational model has nothing to say about operations;
- inheritance;
- computationally complete languages, while the SQL relational language is not computationally complete.

This additional power provides substantial productivity benefits for software developers. This additional power and complexity, however, has its price. Compiler systems understand the effects of a relational query more easily than they understand an arbitrary C++ program. But something similar to the OODBMS approach is required when the application demands more power than the relational model can supply.

Implications for Managers

The most important consequence of the *Manifesto* is that it represents a convergence of research opinion on the proper direction for OODBMS technology. It is, of course, not the role of academic authorities to lead standardization efforts; that role is better filled by industry groups, such as the Object Management Group which will be featured next month in this column. Nonetheless, the work of the industry is made easier and more predictable by the convergence of research opinion.

The Manifesto, however, is not a panacea. The rules are broad, and market reality is such that all companies offering an OODBM will state unequivocally that they satisfy the dictates of the *Manifesto.* This is not bad. OODBMS technology is still young and in a period of innovation and creativity. One should remember that Oracle, Ingress, and DB2 were on the market from three to five years before ANSI SQL was officially published. This period was essential in the development of that generation. The *Manifesto* is similar to Codd's work in that it establishes a framework for a next generation technology.

We recommend that managers, with a serious interest in evaluating OODBMS alternatives get a copy of the *Manifesto,* and study it with their technical people. The *Manifesto* represents an excellent, first checklist of capabilities. From this a manager can make his or her checklist of important items, according to the company's perspective, for an OODBMS. Also to be considered are platform requirements, support for specific languages, and the need for high-productivity programming tools.

Tom Atwood, *founder and chairman of Object Design, has pioneered the commercial development of object-oriented technology. Previously, he was one of the founders of Ontologic, one of the first companies to introduce a commercial object-oriented DBMS. He has worked extensively in the field of database development at Data General and Digital Equipment Corporation.*

Languages and Environments

Which Language to Choose, Revisited

The first issue of HOOT featured an article on choosing an object-oriented language. Which language to choose remains one of the most controversial questions facing anyone embarking into the world of objects.

The following article is by Bjarne Stroustrup of AT&T Bell Labs who is the developer of the C++ language. To quote Bjarne "The original version of this note first appeared on usenet. It was prompted by a polemic blast sent in reply to a fairly naive question from a novice user. It is a plea for sanity and rationality in matters of languages and systems."

Following the original posting of the note, a panel was held at OOPSLA '89 on this topic. Dan Bobrow (from PARC) chaired the panel in which each of the three language developers presented what they appreciated about the other's languages:

Gregor Kiczales (PARC, developer of CLOS) presented Smalltalk,
Peter Deutsch (ParcPlace, developer of Smalltalk) presented C++,
Bjarne Stroustrup (AT&T, developer of C++) presented CLOS.

Despite the fact that this panel was held at OOPSLA'89 as the last session on Friday afternoon, the discussion was well attended and well received. All three of the panelists presented essentially the same message, as delivered by Bjarne below, while also demonstrating their own biases, as one would expect. A summary of the messages would be: "Use the language that fits your needs!"

Jeff McKenna

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes,*** I want to be plugged into the latest thinking and developments in the object-oriented technology. Immediately enter me as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	2 years (24 issues)
☐ ~~299~~ $249	☐ ~~$598~~ $498

outside the U.S.—add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT—U.S.currency drawn on U.S. bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card #_____ Exp. date_____
Signature _____

Name _____
Company _____
Address _____
City _____ State_____ Zip_____
Phone_____

For immediate service—call the Hotline 212-972-7055 or Fax 212-370-4983

HOTLINE on
Object-Oriented Technology
The International Newsletter

310 Madison Avenue
Suite 503
New York, NY 10017

BULK RATE
U.S. POSTAGE
PAID
PERMIT No. 47
MENASHA, WI

HOTLINE on
Object-Oriented Technology
The International Newsletter

February, 1990 **Volume 1, Number 4**

GREED: A Complete Example

by Tom Love

For years I have felt that the object-oriented community needed a good example program to help explain objects and I have spent several years looking for such a program. Good example programs, however, are hard to find . You be the judge; I think I have finally found one.

The example I found is a game program, which involves physical modeling, rules, graphics, an iconic user interface, and report generation. The game was introduced to me by Capers Jones; I do not know from where it originated. The game is Greed.

To implement the game in all of the major object-oriented programming languages, we held a competition at OOPSLA '89. The rules of the Greed exercise were as follows:

1. OrgWare and Knowledge Systems Corporation would provide all participants with the same written specification, by fax, several weeks prior to OOPSLA '89.
2. Any errors in the specification, or misunderstandings arising from the specification, would be corrected by sending identical clarifications to all participants.
3. All implementors would be invited to present their results at a special session during one evening of the OOPSLA conference.
4. Everyone who made a presentation and submitted an implementation would be provided with a copy of all other implementations and a gift (a Greed T-shirt provided by Hewlett-Packard).
5. It was assumed that all implementations were copyrighted material. For further use of these implementations, the permission of each author must be obtained.
6. It was assumed that the implementor was very experienced in the use of his or her chosen language, component library, and development environment.

Before going further, I wish to thank Reed Philips, and the wonderful staff at Knowledge Systems Corporation in Cary, North Carolina, for their participation and collaboration . We could not have completed this project without their help. I would also like to thank Hewlett-Packard Corporation for having provided a reward to all participants — the Greed T-shirt. Finally, of course, it was the extraordinary participation by experts from many companies that made this project such a success.

Nearly 250 people attended a three hour session on Tuesday evening of OOPSLA, even though the session was in direct competition with a free bar, and

continued on page 3

Contents

Feature:
GREED: A Complete Example
1

Thoughts From the Editor
2

Databases
OMG: An Industry Push for Standards
6

Project Management
Talking to Project Managers
9

Languages and Environments
OOP in the 1990's
12

Publicity Corner
15

Book Review
Object-Oriented Programming in Turbo Pascal 5.5
16

Product News
17

Company Showcase
18

Thoughts from the Editor

As we edge further into the 1990s, "The Decade of the Object," we continue to bring to you the latest from the movers and shakers in the object-oriented world.

This, the February issue of the *Hotline* begins with a feature article by Tom Love on the Greed programming "competition," which was held at OOPSLA '89. His analysis of the control structure of some of the entries, as well as a presentation of the time required for some of the implementations, is fascinating.

We introduce with this issue columnist Adele Goldberg. Her first column is an exposition on the management issues of object-oriented technology. This article sets the stage for a series of columns, which will prove vital to those concerned with the practical application of objects.

Tom Atwood herein continues his series on OODBs, with a discussion of the Object Management Group and its anticipated role in standardization.

Zack Urlocker reflects in this issue upon the OOT's recent past and he speculates on its future, in an article that is both timely and informative.

We have also included a book review, have more companies to introduce to you and also more new product announcements (Christmas did slow things down!).

We have received a number of calls from readers of HOOT to compliment us (Thanks!) and also to get information from us about various aspects of the technology. Keep those comments coming!

We are looking for a writer to report regularly on the various activities in the 'hybrid' language area of OOT (C++, Object Pascal, Object Cobol, etc). If you are interested or know of a suitable person, please give us a call.

See you next month.

Jeff McKenna
Editor

Letter from the Publisher

HOOT is definitely off and running. Subscriptions are coming in faster than expected—and from all around the world. Though only its fourth issue, HOOT is pleased to have already published pieces from several of this technology's leading thinkers—Goldberg, Stroustrup, Atwood, Love and others.

In efforts to "cover the waterfront" our editorial team is scanning all of the software trade magazines for anything mentioning "object-oriented." Our search is finding 110-135 published articles a month dealing with some aspect of products and technology. We serve as your "clipping service" as we report the most significant findings in our *Publicity Corner* column.

As the technology grows in popularity, an increasing number of conferences devoted exclusively or in part to OOP appear. This year, Apple's MacApp programming environment has its own conference in San Diego (Feb. 5-9). And USENIX has its second C++ Conference in San Francisco (April 9-11). At Software Development '90, a 4 day conference in Oakland featuring a smorgasbord of 183 sessions on design, languages, management and development, I count approximately 50 classes devoted all or in part to O-O technology. And SCOOP-East is scheduled again in Boston on May 21-23. You can rely on the HOOT calendar to keep you informed on all upcoming OOP related conferences and events.

Enjoy the issue.

Richard P. Friedman
Group Publisher

HOTLINE℠
Object-Oriented Technology
The International Newsletter

Jeff McKenna
EDITOR

ADVISORY BOARD
Jim Anderson
Digitalk, Inc.
Tom Atwood
Object Design, Inc.
Grady Booch
Rational
Larry Constantine
Consultant
Cort DeVoe
IBM
Adele Goldberg
ParcPlace Systems
Bertrand Meyer
Interactive Software Engineering
Reed Phillips
Knowledge Systems Corp.
Trygve Reenskaug
Taskon A/S
Bjarne Stroustrup
AT&T Labs
Dave Thomas
Object Technology International
Zack Urlocker
The Whitewater Group
Eugene Wang
Borland International

Richard P. Friedman
Group Publisher

Xavier Yaffar
Technical Advisor

Melanie Younossi
Marketing Manager

Lori Auerback
Marketing Assistant

The Hotline on Object-Oriented Technology (ISSN#1044-4319) is published monthly by SIGS Publications Inc., 810 Madison Ave., Suite 503, New York, NY 10017. (212) 972-7055.

©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express persmission from the publisher.

Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30. Send address changes and subscription orders to HOOT, 310 Madison Avenue, Suite 503, NY, NY 10017
Submit editorial correspondance to Editor, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062

mdbs Professional Services offers a full range of training, consulting, designing, and application development services using Object/1 and other software products. Clients include the world's largest corporate, governmental, educational and professional organizations.

For more information contact:
David R. Leininger
mdbs,Inc.
Two Executive Drive
P. O. Box 248
Lafayette, IN 47902
voice: (317) 447-1122
fax: (317) 448-6428

Semaphore Training

Semaphore Training delivers a hands-on, real-world approach to technical education in the areas of Object Oriented Design and C++. The emphasis of Semaphore Training has always been upon high quality instruction from senior software developers, who are currently working in a full-time capacity for industry leaders. Offering a wide range of courses, Semaphore's curriculum includes basic courses such as Object-Oriented Analysis, Concepts, Introduction to C++ and Object-Oriented Design, Advanced C++ and Object-Oriented Design, and more. All of our C++ courses feature the latest version 2.0. We can also customize each of our courses to company specifications.

Depending upon our customers' training needs, we offer on-site courses with customized labs, as well as a design review consulting service, and follow up. Additionally, for individuals seeking quality instruction, we offer open enrollment courses on the east and west coasts of the United States, on a monthly basis.

For more information contact:
Charlotte Richter, Account Manager
Semaphore Training
800 Turnpike Street
North Andover, MA 01845
voice: (508) 794-3366
fax: (508) 794-3427

Symantec Corporation

Symantec Corporation develops and markets award-winning Macintosh development environments for C and Pascal programmers.

THINK C and THINK Pascal take programmers from idea to finished product more quickly than any other development environment on the Macintosh. THINK C and THINK Pascal provide full, object-oriented programming support, and deliver programming power, speed, and flexibility in an easy-to-use environment. Both environments include a source-level debugger that makes debugging fast and easy, and that fully supports objects. THINK C and THINK Pascal were used to create leading applications, such as Aldus Pagemaker, Adobe Illustrator, and Claris's MacWrite II, and are a standard in corporations and universities.

Founded in 1982, Symantec develops additional software products for both the Apple Macintosh and IBM PC computers, including Q&A, Time Line, SQZ!Plus, The Budget Express, GrandView, MORE II, SUM II, and SAM. Symantec is a public company, traded on the NASDAQ National Market System (SYMC).

For more information contact:
Susan Smith
Product Marketing Manager
Symantec Corporation
135 South Road
Bedford, MA 01730
voice: (617) 275-4800
fax: (617) 275-2124

Calendar

The HOOT calendar presents conferences and meetings that are focused exclusively on object-oriented technology. It should be noted that many other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to the Editor, HOOT, 8825 SW. Umatilla St., Tualatin, OR 97062

FEBRUARY 5-9, 1990

MacApp Conference San Diego, Winter '90— Tutorials, workshops, general sessions to be held at Doubletree Hotel in San Diego, CA. Contact: MacApp Developers Assoc., 4327 Rucker Ave., P.O. Box 23, Everett, WA 98206, (206) 252-6946.

FEBRUARY 6-9, 1990

Software Development '90— Oakland, Convention Center. Annual 3½ day educational conference and product exhibition. Keynote—Microsoft's Bill Gates. Contact: Bob Saunders, Software Development Series, 500 Howard St., San Francisco, CA 94105, (415) 995-2471.

MARCH 13-15, 1990

The Object-Oriented Symposium— Presented by Digital Consulting, Inc. and chaired by Larry L. Constantine, in Washington, DC. Repeat of the symposium presented in Dec '89. Contact: DCI, 6 Windsor St., Andover, MA 01810, (508) 470-3880.

APRIL 9-11, 1990

2nd USENIX Conference on C++— San Francisco, CA. Tutorials, technical sessions. Conference chair—Jim Waldo, Apollo Computer. Contact: USENIX Conference Office, 22672 Lambert St, Suite 613, El Toro, CA 92630, (714) 588-8649

MAY 21-23, 1990

SCOOP-East— Wang Institute of Boston University, Boston, MA. Co-sponsored by JOOP and the Wang Institute of Boston University. Tutorials, Panels, Product Presentations and Exhibits. Keynote speaker—Niels A. Jensen, CEO, Jensen & Partners, Int.. Contact: Bob Daniels, Wang Institute of Boston University, 72 Tyng Rd., Tyngsboro, MA 01879, (508) 649-9731

JUNE 6-8, 1990

Eurographics Workshop on Object-Oriented Graphics— Konigswinter, Federal Republic of Germany. Workshops cover standardization and OO languages for graphics. Contact: Marja Hegt, O-O Graphics Workshop, CWI, Kruislaan 413, 1098 SJ. Amsterdam, The Netherlands. Tel. 31-20-592-4058.

Regularly appearing in HOOT, this section introduces you to companies offering various products and services in the object-oriented field. Every company which sells an object-oriented product or service (old or new) will eventually appear in the Company Showcase.

Included here is a brief description of the company and its product or service, atributes, with name and phone listing for further contact. This section is provided as an information service to readers and no endorsements by HOOT are implied. Summaries are provided by the companies. Companies wishing to be included in the Company Showcase should send appropriate information to Jeff McKenna, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062. Tel. (503) 692-1889

CACI

The CACI Products Company is in the business of designing, implementing, marketing, and selling software products.

The company is best known for its widely used SIMSCRIPT II.5 language. Now it has introduced MODSIM II, an object-oriented language based upon MODULA. MODSIM II constructs include single and multiple inheritance, dynamic binding of objects, polymorphism, data abstraction and information hiding.

Two features are found in MODSIM II, which are seldom found in OOP languages. These features are built-in constructs for both simulation and graphics. Graphics, icons and input menus are described without programming through an editor. The MODSIM II compiler is itself written in MODSIM II making it easily portable on most computers.

For more information contact:
CACI
3344 North Torrey Pines Court
La Jolla, CA 92037
voice: (619) 457-9681
fax: (619) 457-1184

Geneva Group International

Geneva Group International, with headquarters in San Francisco, CA, was founded by software executives in 1985. Its function is to provide executive search and organizational growth structure consultation services, specifically to emerging software technology companies in the advanced software applications development arena. Geneva Group International specializes in search and recruitment of Presidential, officer level, product marketing, sales, support, and development management professionals for object-oriented programming environments, Databases, Expert Systems, AI, and CASE markets.

For more information contact:
Igor M. Sill
Geneva Group International
Four Embarcadero Center, 1980
San Francisco, CA 94111
voice: (415) 433-4646
fax: (415) 433-6635
or
Craig Bassin
Geneva Group International
7091 Orchard Lake Road, Suite 260
West Bloomfield, MI 48322
voice: (313) 737-2400
fax: (313) 737-1861

GUIdance Technologies

GUIdance Technologies produces Choreographer, the leading graphic user interface development environment. Since 1984, GUIdance has developed and marketed sophisticated, yet easy-to-use software graphic interface development tools and training materials.

Choreographer vastly simplifies the development of graphic user interfaces for applications under MS Windows®, OS/2 Presentation Manager®, and UNIX® Motif. Development time is reduced up to 90%. Choreographer is a complete industrial strength solution, from initial design through implementation, testing, and shipped run-time.

Beyond simplifying the process of creating screen elements, such as the windows and dialog boxes provided by the low-level GUI standard (e.g., Presentation Manager, Windows, Motif), Choreographer provides classes and tools to create application-specific screen elements within the client area of a window. These elements can be moused upon, dragged around on the screen, connected with wires, and can send and receive messages in response to mouse actions.

Choreographer is an object-oriented development environment with a graphical class browser, instance browser, break package, display editor, bitmap editor, text editor, inspector, and complete encapsulation of standard Presentation Manager, Windows, and Motif screen elements.

Choreographer produces standard run-time modules, which can both be called from code written in conventional languages (e.g., C), and can call out code written in conventional languages.

For more information contact:
Linda George
GUIdance Technologies
800 Vinial Street
Pittsburgh, PA 15212
voice: (412) 231-1300
fax: (412) 231-2076

mdbs, Inc.

Since 1979, **mdbs** has been providing professional programmers with software of the highest quality. Our reputation in database management, information management, and expert systems is well known. MDBS IV, KnowledgeMan/2, and GURU have great impact upon the mission-critical applications of large organizations worldwide.

In 1989, **mdbs** introduced Object/1, the premier tool for developing applications in OS/2 Presentation Manager. Graphical user interface design and construction is accomplished in minutes, with access to multiple database management systems provided through class libraries. The underlying object-oriented programming system, with a C++-like syntax, offers hundreds of classes and thousands of methods in source code form.

continued from page 1

the session had little public announcement. Each implementor had the opportunity to describe his own implementation, and at the end of the session, most implementors did live demonstrations. I was very pleased that there was not a single "my language is better than your language" comment during the three hour session. Instead, participants sincerely wanted to understand the experiences that other competent implementors had had using their respective languages and development environments.

Greed Specification

The following specification was provided to all participants by fax:

> Greed is a dice game played by two or more players. The object of the game is to tally points from the rolls of the die,and to be the first player to score 5000 points. There are five die in the game, which are rolled from a cup.
> To enter the game, a player must score at least 300 points on the first role of his turn, otherwise the player is considered "bust." If he goes "bust," he must wait until his turn to role again. If his first roll does produce 300 or more points, the player then has the option of stopping, thus keeping the initial score, or continuing. To continue, the player rolls only the die that have not yet scored in his turn. A player may continue rolling until all the dice have scored, or until he is "bust." With the exception of the entry roll, a "bust" is when an individual roll produces no points. The player may stop and keep his score after any roll, as long as he is not "bust."
>
> Each roll of the dice is tallied as follows:
>
> - Threes of a kind score 100 x face value of one of the three die. If the three of a kind is 1s, then it is scored as 1000. 22234 = 200 points; 43414 = 400 + 100 = 500 points.
> - Single 1s and 5s score 100 and 50 points, respectively.
>
> Examples (for first roll):
> 44446 = 400 points, and the player would have the option to role the 4 & 6.
> 11111 = 1000 + 100 + 100 = 1200 points, and the player has the option to roll all five die again.
> 12315 = 100 + 0 + 0 + 100 + 50 points, and the player would have the option to role the 2 & 3.
>
> Finally, the winner is determined after a player collects a total score of 5000 or more, and all players have had an equal number of turns. If, for example, a player scores over 5000 points, he may still lose if a subsequent player ends up with a final score greater than his.

In addition, several suggestions were provided concerning possible objects to be included in the final implementation. These suggestions were provided in order to facilitate later comparison, among the implementations.

Many strategies are possible and the game is more interesting than it may appear at first. The game can be played by anyone capable of rolling the dice and learning even a rudimentary strategy. Gayle, my four year old daughter, chooses to roll until she either busts or has only one die left to roll. She wins surprisingly often.

Like any attempt to write a specification, this first attempt was not perfect. It was incomplete, in spite of the fact that it had been tested with a trial implementation prior to sending it out to all the implementors. Several days after the initial specification was sent out, a modified one was sent out with corrections. What is shown above is actually the corrected specification. The changes were so minor that none of the implementors reported any difficulty in coping with the differences.

The Designs

We were trying intentionally to receive similar implementations of the same program in each of the object-oriented languages. The similarity of the programs, indeed, was striking . The second most striking aspect of the design process was the way in which each participant chose to elaborate on the design based upon his own interests. Those interested in rule-based programming elaborated upon the strategy. Those interested in graphics were completely carried away with animation, and several people worked to show off the special strengths of their particular language, development environment, or development tools.

The Implementations

A detailed analysis of all the implementations that were done could fill one, if not several, Ph.D. dissertations. Space permits only a brief summary of the results of this exercise. I intend to do a much more comprehensive analysis with complete source code examples, in my forthcoming book, Object Lessons, to be published by Addison-Wesley.

Two implementors, Ward and Karen Cunningham, provided an especially interesting set of examples, because they did the design using CRC cards.They then did implementations in HyperCard, C++, and Smalltalk-80.

Basic

At OOPSLA '89 I was given an article from the May 1987 issue of Laptop User Magazine, which described a Greed implementation in Basic for a laptop computer. That implementation provided a wonderful example of how not to write

software. It violated almost every admonition of code writing procedure presented in Elements of Programming Style—e.g., gotos, no comments, complex control flow, and complex formatting. I even found typing in the program to be amazingly difficult. A sample is worth seeing;

```
220   IF X + Y < 500
          THEN BEEP
               GOTO 110
225   IF O > 0 OR T > 0 OR F > 0
          THEN BEEP
               PRINT @206, "Are You Sure Y/N ";
               E$ = INPUT$ (1)
               IF E$ = "Y" OR E$ = "y"
                    THEN 230
                    ELSE PRINT @206, SPACE$(16)
                         GOTO 110
230   IF S>0 THEN Y = Y + S
               PRINT @235, USING "#####"; Y
               GOSUB 505
               GOSUB 515
               GOTO 235
          ELSE 110
```

This example has been carefully reformatted to make it more comprehensible than the published code, which looked like this:

```
230   IF S>0 THEN Y = Y + S: PRINT
      @235, USING "#####"; Y: GOSUB
      505: GOSUB 515:GOTO 235:
      ELSE 110
```

As object-oriented programmers, we should never forget that most of the world's software is written as the above published code. As we attempt to explain to the world why objects help, keep line 230 in mind!

In spite of its total intractability, I feel sure that this implementation wins the prize for the implementation that runs at the highest speed in the smallest machine. Even as such, the implementation could have been made much more comprehensible.

Capers Jones provided me with another Greed implementation in Basic, which can be thought of as competent, structured basic. I have not had a chance to analyze the implementation, other than to say it is much more legible than the laptop Greed.

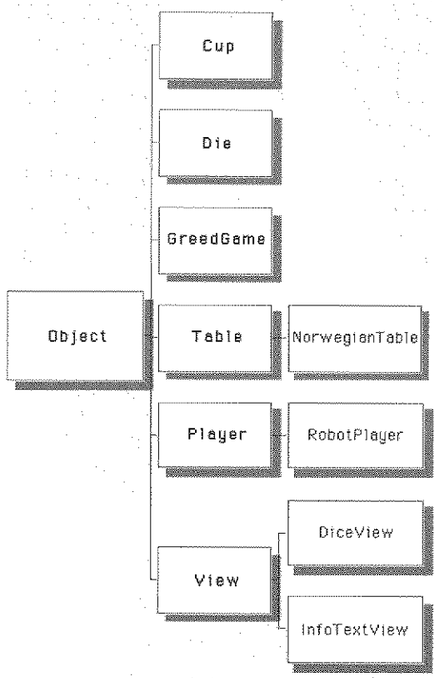
Smalltalk

Four implementations were presented in Smalltalk 2 from Norway (Else Nordhagen and Trygve Reenskaug), one from North Carolina (KSC), and one from Oregon (Ward Cunningham).

Trygve did a marvelously literate design for the game using some new tools that he has developed for doing simultaneous design, documentation, and development. As he described it, the design involves a combination of "design with documentation, implementation with documentation , testing with documentation and exploratory programming with testing , reverse engineering with documentation and testing with documentation."

His result after 27 hours of work was a very literate design, but he go so involved in the design and documentation that he did not complete the implementation!

By contrast his colleague, Else, spent 17 hours to produce a working prototype involving 9 new classes and 132 methods. It was sparsely documented, but was such a clean design that it was easy to understand. Her basic design follows:



Each Greed game knows its players, its table, its cup, and the player who is currently active. The players know their names, scores, the total score, and whether or not the game is in the first round. The table object contains the rules for the game. Subclasses, such as NorwegianTable, contain the rules for a variant of the game known in Norway. The die have values and are able to spin. The RobotPlayer plays based upon a hardwired strategy, which can be changed by altering various methods known to this object.

This implementation in Smalltalk involved 825 lines of Smalltalk code. Interestingly, 509 changes were made to correct errors after the initial prototype was working—and it still only took 17 hours! Thank goodness for efficient development environments.

C++

Karen Cunningham did a C++ implementation of the CRC design that she and Ward created together. This imple-

Product Announcements should be sent to the Editor, HOOT, 8825 SW Umatilla St., Tualatin, OR 97062.

Symantec Announces THINK C, Version 4.0 with Object-Oriented Programming Support

Symantec Corporation announced that THINK C™ version 4.0 (formerly THINK's Lightspeed C), the first Macintosh® compatible programming environment that supports object-oriented extensions to C, is now available. THINK C™ version 4.0 provides two essential elements for effective object-oriented programming: full source-level debugger support for objects, and a class library.

The THINK Class Library provides code for all the standard Macintosh user interface components and behaviors, such as windows and scrolling, thus eliminating the need for developers to recreate common code. Symantec is the only vendor that provides a comprehensive solution to object-oriented programming for both the C and Pascal (THINK's Lightspeed Pascal™) languages. Thus programmers in either C or Pascal can reap the benefits of object support.

Additional new features in version 4.0 include full standard ANSI libraries and enhanced ANSI language conformity for writing portable code. Other enhancements in code resource support make it easier to develop resources, and give the option to write multisegments resources. The advanced, in-line assembler now supports instructions for the 68020 and 68881, and includes new optimizations for more efficient code.

"With THINK C version 4.0, Symantec is at the forefront of the drive to provide complete object-oriented programming support on the Macintosh," said Gordon E. Eubanks, Jr., president and CEO of Symantec. "As the leading C vendor on the Macintosh, Symantec is committed to responding to customer feedback and improving programming technology by providing object-oriented programming and ANSI compatibility in THINK C."

THINK C version 4.0 runs on a Macintosh, with one megabyte of RAM. To use the source-level debugger, two megabytes of RAM and Apple's Multifinder™ are required.

The suggested retail price of THINK C version 4.0 is $249. Registered owners can upgrade to version 4.0 for $69, plus shipping and handling. For further upgrade information, users can contact Symantec's Customer Service Department at (800) 441-7234 (U.S.), or (800) 626-8847 (CA).

For more information contact:
Symantec Corporation
10201 Torre Avenue
Cupertino, CA 95014-2132

Oasys Announces Green Hills C++ Compiler

Oasys, Inc. has announced the Green Hills C++ compiler, the only C++ compiler to support cross and native mode development. Green Hills C++ was developed by Green Hills Software, specialists in optimizing compilers. It is fully integrated with the Oasys 680x0 and 88000 Cross Tool Kits, which enables embedded systems developers to take advantage of object-oriented techniques. A high performance C++ compiler in the native and cross environments, Green Hills C++ supports Kernighan and Ritchie C, and compiles to the ANSI C standard.

Oasys, a Green Hills company, is the largest supplier of C++ software to the commercial market, with 3 years of experience in supporting more than 2000 customers who are implementing C++ applications. Oasys is also well established in the embedded systems market, offering a complete set of cross development tools for M680x0, M88000, Intel 80x86, and i860 target environments, which include the Green Hills compilers. "Now with Green Hills C++, embedded systems developers have a language solution, which combines high performance, availability across hosts, and OOP support," said Michael Olfe, General Manager of Oasys.

Green Hills C++ provides object-oriented programming features such as data abstraction, strong type checking, and overloading of function names and operators. New C++ features include classes with scope, and overloading new and arrow operators. Green Hills C++ also includes compiler optimizing techniques such as inlining, loop unrolling, and register caching. "We understand the importance of optimizing code—that's where the Green Hills expertise really pays off," said Olfe.

End-users will find that the Green Hills C++ compiler will reduce the development cycle by taking advantage of existing class libraries, and the compiler will cut down on maintenance costs by producing more reusable code. Green Hills C++ is based on proven technology that produces tighter code, which is easier to understand and to maintain. The inclusion of validated Kernighan and Ritchie C, and ANSI C allows developers to take existing C, code and enhance applications, thus allowing their applications to evolve rather than having to rewrite them to implement OOP.

Green Hills C++ compiler is available from Oasys on the Sun-3. Designed for portability, the compiler will be ported to other UNIX workstation and minicomputers soon. Call Oasys for availability and pricing.

Oasys will continue to support Designer C++, the C++ translator developed by Glockenspiel, Ltd. OASYS will provide current customers with the ability to upgrade to the Green Hills C++ compiler. Oasys offers over 120 tools aimed at software development on minicomputers, workstations, and mainframe systems using UNIX, VMS, and other operating environments. OASYS's products, many developed through OEM, VAR, and other exclusive distribution relationships, are available for cross and native development, with full documentation and technical support.

For more information, contact:
Ann Bischoff, Director of Marketing
Oasys, Inc.
230 Second Avenue
Waltham, MA 02154
voice: (617) 890-7889
fax: (617) 890-4644

Oregon C++ Ported to NCR Tower 32 Series

NCR Corporation, Europe Group, and Oregon Software, Inc., have signed an agreement to port Oregon C++ to the NCR Tower 32 Series.

Oregon C++ is actually three compilers in one software development system—a C++, an ANSI C, and a K&R C compiler, and is also a symbolic debugger for debugging programs written in any one of the three C language options.

Oregon C++ was chosen by NCR Corporation's Europe Group because it met their Netherlands' customer's requirement for a true C++ compiler and a source-level debugger. NCR Corporation will refer customers to Oregon Software, Inc., and its distributors, for purchase of the software. The U.S. list price is $3,000 for a single-user license.

Oregon Software, Inc, was founded in 1977, and is known world-wide for its Pascal-2 compilers. In addition to Oregon C++ and Pascal-2, the company has also developed Oregon Modula-2. The Oregon Software language compilers are available on a variety of UNIX workstations, as well as VAX/VMS. Cross-compilers are also available.

For more information, contact:
Oregon Software, Inc.
6915 SW Macadam Ave., Suite 200
Portland, OR 97219
voice: (503) 245-2202
fax: (503) 245-8449

Object Oriented Programming in Turbo Pascal 5.5

(Addison-Wesley, 1989)

by Ben Ezzell

Tens of thousands of satisfied Turbo Pascal users are about to discover that the latest version of Turbo Pascal supports object-oriented programming. A large user community, such as the users of Turbo Pascal, is likely to have a much greater impact on the adoption of object-oriented programming than other language communities would have.

My experience in teaching college level "computer science" classes made clear to me the impact Turbo Pascal has already had. Ten years ago, most students who came into an introductory programming class with previous experience had used BASIC. At that time they had trouble accepting the idea that structured programming was a better way to do things. When I more recently taught an introductory programming class, most students with previous experience had alreadyused Turbo Pascal. They accepted structured programming, not only as the best choice, but as the only choice.

Turbo Pascal 5.5 users will need reference material to learn the "what" and "why" of object-oriented programming. Ben Ezzell's book, Object-Oriented Programming in Turbo Pascal 5.5, is an early entry into this reference market. Unfortunately, the book falls somewhat short of its goal of teaching object-oriented programming to Turbo Pascal users.

Ezzell's volume is a series of examples, which demonstrate object-oriented concepts. The series is based upon two main examples: graphical user interface components (scroll bars, buttons, etc.), and a telephone directory. The programs are available on disk from the publisher. A large percentage of the text of the book consists of code listings and detailed explanations of how the code works.

Ezzell assumes the reader has knowledge of Turbo Pascal and, to some extent, of version 5.5. There is no language overview, nor summary of the new features, and there is a heavy reliance upon "Turboisms" in the examples. Several chapters provide a good treatment of pointers and dynamic allocation, a topic not frequently covered adequately in Pascal language references.

The book must be judged by how well it demonstrates object-oriented programming, and its elegance, to the intended audience. This audience can not be assumed to know the "what" and "why" of object-oriented programming. There is, however, no introductory section to define object-oriented programming, nor to explain why it is important in the process of software production. In fact, in one example, the author chooses not to use object-oriented programming "to avoid complicating the utility." This clearly gives the reader the wrong impression about object-oriented programming!

Generally, a language is considered to be object-oriented if it meets three criteria: objects are organized into classes, classes are organized into hierarchies using inheritance, and message sends are polymorphic. A demonstration of object-oriented programming must include a discussion of all three of these concepts. Object-Oriented Programming in Turbo Pascal 5.5 covers the three criteria with varying degrees of success.

Ezzell does a good job of explaining data abstraction. He drives home the point that objects contain data and functionality, and explains why (and how) the data in objects should be accessed only through methods provided by the object. The distinction between object types and instances (to use Turbo terminology) is less clear.

Inheritance is a construct that can be easily abused. Object-Oriented Programming in Turbo Pascal 5.5 includes examples that demonstrate a less than ideal use of inheritance. Programs can use inheritance for ease of implementation, or actually to take advantage of the "kind-of" relationship (for example a Cat is a kind of an Animal and therefore can be derived from it). Most examples in this book use inheritance simply to save on code in the derived class. For example, Square is derived from Circle—one could never call a Square a kind of a Circle.

Polymorphism is the most elegant and important facet of object-oriented programming. Ezzell begins by comparing static and dynamic binding of calls to methods in a class hierarchy, from within other methods in the hierarchy. He explains this very well with several good examples. But Ezzell's explanations and examples of sending messages to objects of an unknown type (an extremely important concept) are sorely lacking; he includes one example of a message send to a dynamically created object from a linked list.

Object-Oriented Programming in Turbo Pascal 5.5 is not the perfect introduction to object-oriented programming. Despite this,its early entry on the market will probably ensure its placement on many bookshelves. It will be helpful for its intended audience, especially given the inclusion of debugged examples with which to experiment. This book, however, should not be the only source of information for anyone learning either object-oriented programming or Turbo Pascal.

Reviewed by Karen Ryer Cunningham, a consultant specializing in object-oriented training and software development. (503) 626-4119

mentation required 455 lines of code, excluding blanks and comments. The implementation, like the two Smalltalk implementations done by the Cunninghams, involved a very simple teletype user interface—quite unlike the user interfaces developed by Ms. Nordhagen in Smalltalk-80, by Dr. Cox in ProGraph, or by the Apple team in Object Pascal.

HyperCard

Ward and Karen did a wonderful implementation in HyperCard, which not only played the game, but also provided an image of the current player, voice interaction, and a sound and graphic display of the rolling dice. The design and implementation required only 6 hours.

Their conclusion was that, "Smalltalk no longer deserves its reputation as the best tool for experimenting with user interfaces. HyperCard is clearly better."

ProGraph

Dr. Phil Cox, of Gunakara Sun Systems, provided an intriguing implementation of the Greed game in ProGraph. Even more interesting was his analysis of the strategy. He has analyzed it as only a competent mathematician would:

"At a particular point during a turn it is necessary to decide whether or not to continue rolling. To do this we need the following definitions.

```
Let n =  number of dice in the game
    k =  number of dice in next roll (1 ≤ k ≤ n)
  S (k)= expected score from rolling k dice, including
         scores from subsequent rolls (1 ≤ k ≤ n)
    A =  the current maximum score among
         preceding players in this round
    N =  the current maximum score among
         following players in this round
    C =  my current score before this turn
    T =  my score in this turn so far
 q(X) =  the probability of scoring at least X in
         one turn, for any nonnegative integer X
   bk =  probability of a bust in a roll of k
         dice
```

(a) Now if A ≥ 5000 and T+C ≥ A, I should roll, since this is the last round of the game and someone else has the winning score.

(b) If T+C ≥ 5000, T+C > A, and I am the last player in the round, I should not roll.

(c) If T+C ≥ 5000, T+C > A, and there are players left in this round, then this could be the last round of the game, and I should worry only about the players that follow. Obviously T+C > N, since otherwise N ≥ 5000, and the game would have ended in the last round. So T+C-N is a positive integer.

If q(T+C-N) > bk I should roll. That is, I should roll if the following player with the highest score has a greater chance of getting enough points to beat me than I have of going bust on the next roll.

(d) In all other cases, if (1 - bk)T + S(k) > T, I should roll. That is, I should roll if the expected turn score from rolling is greater than the current turn score. We can simplify the comparison as follows: I should roll if S(k)/T >bk."

The above was an excellent example of mathematical formalism adding some real value during the specification and design of a computer program. Often such formalism is counterproductive.

ProGraph is a visual programming language for doing object-oriented development. As such, it was a challenge to do any direct quantitative comparison with textual languages. But it was significant that ProGraph was sufficient to develop this example program with very modest effort.

While I find the ProGraph development environment quite attractive and comprehensible, it was interesting to see how the archive output looks on paper. The output seemed almost as obscure as the unstructured Basic in this form. More work on this is required.

Object Pascal

Kurt Schmucker and his Apple Scientific Computing staff decided to do a serious and complete implementation of greed for use as a comprehensive example of Object Pascal and MacApp. It was a real tour de force—color graphics, n die, m sided die, realistic sound, a complete scoring pad, etc. All this capability was fully Mac compatible, including even the ability to establish preferences for each game, and to print results.

This implementation required 179 person hours of effort by a team, and involved 20 classes and 123 methods. Many hundreds of methods and tens of classes were reused.

The implementors did make one mistake. Their final implementation used 10 die rather than the 5 that were clearly called for in the specs! They found that 10 die made for a more interesting game.

Other Implementations

Several other implementations were done, including Eiffel, Trellis, Actor, a design in IDE, and Complete C. The authors, however, provided these implementations in a variety of formats and physical media. Work continues in order to get the implementations into the agreed standard form, readable on a Macintosh. Stay tuned for more information and analyse as these implementations become available.

A Quantitative Comparison

As an initial analysis of the various implementations, I counted the words and lines of uncommented code for three implementations, which were very similar in functionality. All comments and blank lines were stripped out in advance. Basic and C++ were roughly comparable in size, and were about twice the size of the Smalltalk implementation.

These three implementations used a character-oriented user interface, much as one would expect to be used on a teletype that was connected to a timeshared computer.

	words	lines	branching stmts	% branches	looping stmts	% looping
Basic	1698	470	211	45%	31	7%
ST-80	720	250	24	10%	58	23%
C++	904	455	33	7%	16	4%

While the size of the Basic implementation and the C++ implementation were roughly comparable, they were very different in complexity. A staggering 45% of the statements in the Basic program were branching statements, compared with only 7% in C++. Of the 211 branching statements, 43 were unconditional GOTOs.

While I have written some fairly complex Basic programs to control laboratory experiments in the early 1970s, I admit that I found the basic program totally intractable after hours of study. By contrast, the Smalltalk and Object Pascal programs were totally transparent, and were extremely easy to understand. C++ was not very difficult to understand, but was significantly more complicated than Smalltalk and "devoid of ergonomics."

The existence of appropriate class libraries directly affected the user-interface design for the participants. Languages like Object Pascal, Smalltalk, and HyperCard had very pleasant, visual user interfaces. Languages like Basic and C++ presented text-oriented interfaces.

While too much should not be made about development time statistics,due to a total lack of control over individual differences, some observations are worth noting. It took 10 times longer for the Apple team to develop an application with 123 methods than it did for a solitary Smalltalk developer to develop one with 134 methods—179 hours versus 17.

One of the most sophisticated graphical interfaces that involved speech was also the fastest to develop; it took only 6 hours. It took more than three times as long to do a comparable job in Smalltalk. A conclusion might be that "less language can be more productive."

What has not been evaluated as yet is run time performance or memory utilization. We can guess that there will be much variation among the respective implementations.

A Qualitative Comparison

Since the Cunninghams implemented the same design in four different languages, they were able to compare directly the benefits and disadvantages of what it felt like in each. They concluded that "HyperCard is quick and easy, but not best for complex problems; Smalltalk is easy too, and has the best code, but it no longer deserves its reputation for providing the best user interface; C++ is difficult, even for small tasks and might be best used for big systems."

It is unclear to me why something difficult, even for small tasks, might be better for "big systems." It is my experience that the larger the system, the more important it is for the design and implementation to be easy to understand, easy to modify, and easy to use. Remember, the larger the system, the lower the average quality of developer who is likely to be involved in its development and maintenance.

All of the object-oriented languages, however, were a dramatic step ahead of unstructured procedural languages!

Tom Love *is an object-oriented pioneer, and a long time practitioner, who was involved as a founder of Stepstone Corporation. He currently provides management consulting services to organizations investigating the technology. He may be reached at OrgWare, Inc., Roxbury, CT, (203) 350-4331.*

Databases

OMG: An Industry Push for Standards

by Tom Atwood

One of the biggest problems with object-orientation is that it is difficult to categorize. Fundamentally, object-orientation is a conceptual approach to problem solving. When applied to software development, OO provides a better way to engineer various systems, be they user interfaces, languages, or DBMSs. The broad range of object-oriented applications does, however, lead to different interpretations of the technology. Despite the confusion, object-orientation is being heralded by many as the dominant computing paradigm of the 1990s.

The excitement surrounding object-orientation has spawned the Object Management Group (OMG), an international computer industry consortium. The OMG is dedicated to promulgating standards in object-oriented application development, and execution environments. Member companies, from the U.S., Europe and Japan, include Hewlett-Packard, Data General, Prime/CV, Sun, Unisys, Canon, and Nixdorf. It is my understanding that active discussions are underway with Digital, Microsoft, and IBM.

I asked Richard Soley, of OMG, and Dave Hentchel, who works for Prime/CV, and who is the leader of the OMG's Persistent Object Storage Task Force, to help explain the role of the OMG in general. They described the organization

Publicity Corner

This month our reviewers gathered approximately 135 articles, gleaned from the nation's press, containing info on OOT. The distribution of the articles was similar to the distribution of previous months, with about 45% of the articles being announcements of OO products. Some 35% of the articles were of a general nature, with more and more examples of OO programming (how to articles) for readers to consider. The remaining articles had topics that were scattered over various areas, with far less use of the term "object-oriented" in reference to iconic user interfaces.

CHRISTOPHER STONE of the Object Management Group, was highly visible in the news this month. He was the author of an article on the roles of standards in the development of object-oriented technology, and the anticipated role of OMG, in the November issue of *UnixWorld*. He states, "Creating standards before the ubiquitous word "de facto" creeps into the vernacular, is the primary mission of OMG." In another article, he is quoted as saying, "You will be able to go into a computer store and buy objects just like you go into an automotive store and buy air filters" (Nov 20, *Computer System News).*

●

PCWEEK (Dec 11) reported that the CODASYL November meeting received three proposals on object-oriented extensions to COBOL: one each from Hewlett-Packard Co., Micro Focus, Inc., and Realia, Inc. "While members of the task force disagreed about the specific implementation of the OOP extensions, they said they expect to find common ground without too much difficulty."

●

THE SAME ISSUE of *PCWeek* contained a lengthy review of five OS/2 PM "Developer Kits" that are all touted as being able to reduce the effort required to develop applications in OS/2 PM. Three of these were object-oriented: Smalltalk V/PM, Choreographer 1.0, and Object/1 1.0. The other two, CASE:PM 1.0 and Applications Manager 2.11, were AI based. The product review contained quotes from individual developers who have used the kits for major development. Each product was scored on the quality of the included libraries, the quality of the OS/2 integration, the quality of the documentation, the usefulness in preparing user programs, and the performance of the subroutines. Smalltalk V/PM scored highest, with a weighted score of 8.4. Choreographer, Applications Manager, and Object/1 followed, each with a score of 7.8, and CASE:PM rounded out the field with a score of 7.6.

The growing interest of government agencies in object-oriented technology was discussed by Si Lyle (president of Ontologic) in an article in the November issue of *UnixWorld*. "... several object-oriented standards should emerge from the Defense Department's Computer-Aided Acquisition and Logistics Support (CALS) program," he said.

●

A COMPREHENSIVE VIEW of the recent OOPSLA '89 conference, by Michael Swaine, was presented in the December issue of *Dr. Dobbs Journal*. He covered the conference well, and included a heathy dose of skepticism. "If you believe Johnny Carson, some 20 percent of the people in this country examine their belly buttons daily. A disproportionate number of these navel observers were at the Hyatt Regency in New Orleans, asking themselves, and each other, what object-oriented programming is. To be fair, I should add that just as many people were answering the question as asking it; I just wish I could say that they were giving the same answer." He also notes one of the current, relevant questions regarding OOP: " Can an OOP system provide the strong typing needed for software engineering, and at the same time remain a good system for rapid prototyping?"

●

IN ADDITION to the above article, the December issue of *Dr. Dobbs Journal* included an interesting article, which questioned some of the claims of object-oriented programming. The author, Scott Guthery, first inquires about the evidence: "To my knowledge, there has been absolutely no evidence gathered, or experiments performed, to validate the claims made for object-oriented programming, particularly for object-oriented programming in the large." Guthery also asked for the definition of an object: "The IEEE 'Tutorial on Object-Oriented Computing'... contain(s) many definitions and descriptions of an object. These definitions come in two basic flavors. One flavor talks about modeling reality and the other talks about encapsulated collections of programming tricks. Stripped of its fancy jargon, an object is a lexically-scoped subroutine with multiple entry points and persistent state." He goes on to question the reusability claims, points out that no program is ever fast enough so why should one make it harder on oneself, and asks about the mixing of OO languages with other languages. He then questions whether OOP will ever be used in a production system. He concludes: "Object-oriented programming runs counter to much prevailing programming practice and experience: in allocating and controlling software costs, in modularity, in persistent state, in reuse, in interoperability, and in the separation of data and program. Running counter to prevailing wisdom does not, of course, automatically make an innovation suspect, but neither does it automatically recommend it. To date, in my opinion, advocates of object-oriented programming have not provided us with either the qualitative arguments or the quantitative evidence we need to discard the lessons painfully learned during the first 40 years of programming."

●

AN ARTICLE by Jeff Moad in the November issue of *Datamation* examines the barriers to the acceptance of object-oriented technology in organizations. He does this by reporting on the activities of various companies in the northwest. He states, ".. some of the strongest advocates of software reuse are beginning to sound a warning. Users shouldn't think of reusable technologies, such as object-oriented programming, as a magic genie that will pop out of a bottle and solve their software problems. Just like anything else, planning and hard work are required to make these technologies pay off." "Retraining procedural programmers in object-oriented programming approaches, that will result in reusable objects, is often surprising difficult, experienced users say." "'You can write terrible code in C++ just as easily as you can write it in C,' says Mary Wells of Tektronix."

gramming and design experts are explaining the way in which OOP fits in with traditional techniques. This happenstance may seem paradoxical, but I view it as part of a wave of acceptance of OOP. Even the most skeptical of programming experts, after examining OOP closely, have found OOP's features (encapsulation, polymorphism, and inheritance) to be incarnations of accepted design strategies. Whether OOP is new or old doesn't matter as long as it increases productivity. More important than all this, the structured programming experts are setting realistic expectations for OOP, and are reminding OOP devotees that they must continue to refine design techniques, and not to treat OOP as a panacea.

As the market becomes more experienced with object-oriented programming, it will become more sophisticated in its purchases. Over time, purchase decisions will be made on the basis of real benefits, and not just on the basis of hype. As a result, products or services that claim "object-orientedness," without real support, will fall by the wayside. It's unfortunate that there are opportunists in the market; the best thing one can do is to research one's purchase decisions carefully.

Is OOP Following the AI Path?

When people think of the growth of OOP, they often compare it to artificial intelligence. AI emerged from the labs of universities and research centers to become one of the most hyped technologies of the eighties. Needless to say, AI never lived up to expectations. Although artificial intelligence techniques are useful for some applications, for many people AI has become a dirty word. Is OOP destined to repeat the history of AI? Absolutely not.

OOP is different from AI in several respects. First of all, object-oriented programming is better defined than artificial intelligence. Artificial intelligence has always been a moving target. Whatever was not understood was called AI; once a problem became understood, it became something else. There may not be a universally accepted definition of OOP, but most experts agree on the basic tenets.

Second, many AI techniques are based on heuristics that are not appropriate for all applications. For example, inference techniques are only suitable for some ill-defined problems. OOP techniques, on the other hand, are much more broadly applicable. They aren't restricted to a single problem domain.

Third, AI has required tremendous investment with very little short-term payoff. Many companies incurred major hardware, software, and training expense to use the new techniques, only to find a relatively small payoff. In order to achieve larger payoffs, applications required more investment.The firms, however, that are pioneering the use of OOP today are finding a two to four-fold increase in productivity. They're getting benefits within months, not years.

The most significant distinction between AI and OOP is that OOP is entering the mainstream. The events of the last year, which I've described above, all bear witness to this. The existence of both hybrid languages and complete OOP languages gives organizations more freedom in integrating OOP into their current projects. Even completely object-oriented languages are easier for traditional programmers to learn than either Lisp or Prolog.

The growth of object-oriented programming should not be compared to the fad of AI, but is more appropriately likened to the revolution of structured programming. The arguments against object-oriented programming sound strangely familiar to anyone who had to defend structured programming in the 1960s. There's a performance penalty for polymorphism, but when used correctly, it is not much more than the penalty of a subroutine call. And, of course, you can write pseudo-object-oriented code in any language if you're disciplined; the only drawback would be that it is about as productive as writing structured code in FORTRAN IV. In another ten years, the debates over "pure" versus "hybrid" object-oriented languages will take on historical charm, much as has the debate about whether the Goto is harmful. The bottom line is that object-oriented programming is here to stay.

Looking Ahead

The history of object-oriented programming is still being written. While there will continue to be technological breakthroughs and improvements in efficiency, I believe that the major changes to come will be more sociological than technical. These changes represent the real hurdles of a new technology. In the early part of the decade of the 90s, more education, better design methods, and greater understanding of how to manage the development of object-oriented programming projects will be more important than language improvements.

As object-oriented programming becomes more widespread, we will witness continuing productivity improvements, due to the broader availability of class libraries. We will also see the development of a common object model and object support at the operating system level. These developments will easily allow us to share objects between languages and applications.

It is tempting to take a "wait-and-see" attitude toward this new technology. Certainly there will be developments in the future that go beyond OOP. I expect, however, that at the end of this decade, object-oriented programming will be as accepted as structured programming is today. By the same token, some new technology will emerge with promises of even greater productivity. There is no doubt that future software revolutions will build on today's object-oriented programming systems.

Zack Urlocker *is manager of developer relations at The Whitewater Group, who are the creators of Actor, an object-oriented language for Microsoft Windows. Mr. Urlocker has taught object-oriented programming to hundreds of professionals, and has written articles for several computer magazines and journals.*

and the work that is being done toward reaching standards in the database management systems area. This description follows.

Object Management Group—The Need

OMG members believe that object-orientation is an excellent approach towards building software that meets customer demands for interoperation of applications in a heterogeneous, multivendor environment. Object-oriented technologies are the key to meeting these demands, provided that standards are in place to protect existing investments in important software systems. A wide range of views, however, exists among members in regard to what constitutes an object-oriented system, and what priorities should be placed on the various parts of the standard. The OMG is striving to come to terms with these differences and to build a usable standard with which future applications can interoperate.

OMG—The Goal

OMG standards have as their ultimate goal:

- To provide portability and interoperability for the applications developer; and
- To create a level playing field for the providers of tools and services.

To make standardization possible in this emerging field, the OMG is constructing a reference model of an object-oriented application development and execution environment.

OMG—Constructing The Reference Model

The operational structure of the OMG includes a Technical Committee, in which all OMG members have voices. Various Task Forces are also part of the operational structure, who are responsible for developing portions of the overall reference model. The procedure for cultivating the various parts of the reference model involves generating a Request for Proposals (RFP) by a Task Force, who wishes to fill in an open part of the reference model. Industry-generated RFPs are evaluated by Task Force members for completeness and interoperability with the overall model. The RFPs are then evaluated by the Technical Committee for inclusion in the OMG core technology.

The OMG is set apart from standards organizations, such as ANSI, by the requirement that industry responses to RFPs must be based upon existing and implemented technology. Rather than construct an academic standard for an object-oriented system, OMG wants to lay the foundation for an interface that can be bought "off the shelf." Future applications developers will be confronted with the need to design an integrated solution that spans a heterogeneous environment, potentially across network gulfs.These architects will be able to code to OMG standards, and will be able to purchase readily an OMG-compliant package from vendors for common functions.

The OMG and Database Standards

How do object-oriented databases fit into database standards? A recent draft of the OMG reference model outlines five major interfaces. The Persistent Object Storage interface correlates directly with object-oriented database development. The interface provides the following functions:

(1) *Storage and retrieval.* Mechanisms for creating, modifying, activating and deactivating objects.
(2) *Name resolution.* Interfaces for identification and location of objects across multiple object stores.
(3) *Repository.* Means of defining, redefining and determining object class structure.
(4) *Transaction management.* Control of versioning and concurrent object access, based on the capabilities of the underlying object store.

The Persistent Object Storage Task Force, responsible for defining the standards and selection of technology for persistent object storage, currently has subtask forces working on five projects. These projects will identify requirements for persistent object storage:

(1) Object Data Model
(2) Model Specification Tools
(3) Semantic Interface
(4) Base Capabilities for Database Interface
(5) Database Coexistence

Persistent Object Storage Task Force—Projects

The *Object Data Model* project will focus on defining common elements of a data model to be used as a base criterion in selection of technologies. This subtask force will specify the minimal requirements for any target database, without placing restrictions on how the data model is implemented.

The *Model Specification Tools* subtask force will select schema specification tools to be used by the task force in

order to present the data model, and other functional requirements. A consistent format for specifying object database technology will speed the process of selecting and evaluating target technologies.

The *Semantic Interfaces* project will define multiple levels of semantic interface for persistent objects, and will specify how these application interfaces should integrate with existing languages and application libraries. Possible options include a function-call interface to retrieve and to store objects, an extension of SQL to accommodate objects, and a class library, defined via object-oriented languages, which provides generic access to multiple databases.

The *Base Capabilities for Database Interface* subtask force is charged with identifying specific requirements for the behaviors that an application should be able to expect from persistent objects. This specification will define the methods and protocols that will be supported, within the scope of the object data model. The specification will also define the capabilities of the semantic interface. This project, too, will set criteria and objectives for evaluating performance for various object persistence schemes.

Finally, the *Database Coexistence* task force will track the requirements and issues specifically involved with migration, conversion, and interoperability of the persistence object interface with existing databases. These databases may use object-oriented, SQL, or proprietary modes of access, and may require creation of run-time libraries to provide some subset of the base capabilities defined above.

It is expected that initial RFPs in these areas will be ready during 1990. Although other OMG technology evaluations will be completed, and subsequent OMG standards promulgated during 1990, it is probable that standards for Persistent Object Storage will not be complete until 1991. After these standards are in place, and compliant products are available, scenarios such as described in the following section will become practical.

OMG—The Vision At Work

Consider an entrepreneurial President, charged with writing a business plan for venture capitalists. He has four vice presidents, each working on a portion of the project. The Technical VP is developing a complete design of the system, with drawings and text stored as object in an OMG-compliant object store. The CFO is developing spending projections; these numbers, and the constraints between them, are maintained in the corporate SQL-language database, which is on a rented mainframe. An OMG-compliant interface is able to extract objects by forwarding SQL queries to that mainframe.

Meanwhile, Sales and Marketing VPs are developing their respective plans for the upcoming product line. Although they are working in separate offices, using different word-processing and spreadsheet software, they share information. Some of this information may be market projections and product identification, shared via OMG-compliant naming and storage/retrieval services.

The President must integrate these disparate sources of information into a complete business plan. Browsing the design document library, using an OMG-compliant application interface, he chooses significant design drawings and text from all four vice presidents. From this he pulls together an executive summary, complete with the product's capabilities, sales projections, market environment, and operating plans.

The President has built a "live document." Rather than being a compendium of the knowledge of his team, he has linked together the crucial parts of the team's work. The applications he uses rely on OMG-compliant Persistent Object Store interfaces to integrate information from a wide variety of hardware and software platforms, without detailed knowledge of the source applications.

This freedom to work without detailed knowledge of particular application architectures, and be insulated from the inconsistencies among those applications, comes from universal compliance with a standard.

OMG—Now

The scenario outlined above cannot occur today. If successful, however, the OMG will help move us to the point where the scenario becomes a reality.

I'd like to thank Richard and David for their contribution. If you would like to contact them, please refer to the phone numbers listed below.

Implications for Managers

I welcome any group whose mission is to advance the cause of object-orientation, although it is still too early to predict the importance of OMG in the industry. There is good news about the OMG, even though several unanswered questions must yet be addressed.

Good News

- The OMG exists early in the development of the object-oriented era, and can help develop early consensus on definitions and standards for object-oriented software, including DBMSs These standards can help facilitate the growth of this technology.

- The OMG was initially founded mostly to promulgate Hewlett-Packard's New Wave. Now, the OMG is finalizing a new charter that establishes its independence.

- As of September 1989, The OMG had a full-time staff. This has spurred its growth, and makes it more likely that the OMG will be able to accomplish significant work in a relatively short period of time.

- The OMG seems to be attracting the major players in hardware and software. This type of membership can only improve the chances that their work will be universally accepted.

This increase in sophistication means that people aren't just curious about OOP, they're using it. One of the most rewarding experiences in speaking at OOP conferences now, as compared with a year or two ago, is that people are coming back after having successfully completed a project. They want to learn even more.

Another trend that signifies increased use of OOP in industry is the response of the academic community in teaching OOP. We've found an increasing number of courses on object-oriented programming in mainstream universities and colleges, not just the research-oriented schools. In many cases, the pioneering is being done by smaller colleges that have the freedom to innovate without seeking the approval of a curriculum committee. The ACM and IEEE have not yet incorporated OOP into the recommended curricula, but it's a safe bet that this will happen within the next two years. After this incorporation, I would expect that all graduates of computer science programs will be acquainted with OOP, thereby increasing the market of OOP programmers.

Watershed Events of 1989

There are several watershed events from the last year that stand out as indicators of a turning point in the history of object-oriented programming. These include the release of the NeXT machine, Microsoft and Borland's support for object-oriented programming, the SCOOP conference held in Boston in June, and the appearance of commercial applications written in object-oriented languages. Prior to these events, even the most ardent supporters of object-oriented programming probably would have expected OOP's popularity to continue at a slow, steady pace. These events, however, propelled OOP into a much more rapid stage of growth.

Although there is great room for debate on the merits of the NeXT machine and its software, the significance of the machine for the OOP market was that it legitimized the technology. Steve Jobs' dazzling public relations skills transformed OOP from an interesting technology into a media buzzword. How many other people could have obtained media coverage for OOP in *Business Week, Fortune* and the *Wall Street Journal?*

Subsequent to this, the simultaneous release of OOP languages by the major PC language vendors, Microsoft and Borland, has brought OOP into the mainstream. Thousands of Pascal programmers are being exposed to object-oriented concepts through Object Pascal's hybrid approach. As these programmers examine Object Pascal, they are also looking into other object-oriented languages, thereby having a catalytic effect on the industry.

The first annual Seminars and Conference on Object-Oriented Programming (SCOOP), held at the Wang Institute of Boston University in June, was the first non-academic conference on object-oriented programming. While OOPSLA remains the forum for technological and research advances in OOP, SCOOP has emerged as a conference for programmers who want to learn and apply object-oriented techniques. SCOOP was bursting to capacity with over 600 attendees who eagerly shared and learned OOP techniques. Scott Knaster, of Apple Computer, called *SCOOP* "the Woodstock of object-oriented programming."

The final watershed event of the year was the release of commercial applications, which were written in object-oriented languages. Although end-users aren't likely to be concerned with the particular language in which an application is written, developers who are hesitant to use OOP, because "it doesn't work for real applications," now have evidence to the contrary.

I've seen a variety of vertical market applications being used in hospitals, information centers, construction companies, and elsewhere, all written in object-oriented languages. Not surprisingly, many of the applications that offer rich graphical user interfaces are written completely in object-oriented languages, such as Smalltalk and Actor. One can even purchase retail software written in OOP languages. Examples of this retail software include dbPublisher (Digital Composition Systems), written in Smalltalk/V, and Whitewater Resource Toolkit (The Whitewater Group), written in Actor. It is not surprising that some of the leading software and hardware vendors are also using OOP for their own commercial products. Index Technologies is using C++ for the next version of its Excelerator CASE product, and Cognos is using Eiffel for a major rewrite of PowerHouse, its fourth generation language.

OOP Products Are Maturing

Most of the popular object-oriented languages have matured significantly in the last year, and there are signs that this maturation will continue in the future. Increased competition in the marketplace will contribute to the continued improvement of OOP languages and tools in the 1990s.

Although memory requirements remain high in most Smalltalk implementations, Smalltalk is now being used for more than just prototyping. Many of the old barriers, such as the use of nonstandard windowing environments, and the inability to communicate with external routines written in other languages, have been reduced.

The standardization of C++ 2.0 is an important event for C programmers, which will increase the availability of compatible compilers, and will eliminate some of the anachronisms of C++. Unfortunately, there are still minor differences between C++ and ANSI C, and no progress has been made on standard libraries.

I expect that there will continue to be improvements in all major OOP languages in 1990. For example, Actor 2.0 will be available by the time this article is published. Actor 2.0 implements its own static object swapping system in order to break through the 640K barrier of DOS, and offers increased compatibility with C. As a result, Actor is better equipped for large applications.

Is OOP a Fad?

In the past year, many have questioned whether or not object-oriented programming is a passing fad. One indicator of the long-term strength of OOP is that structured pro-

- Education of professional personnel about object-oriented technology and practice (learning what is really new, while not forgetting the old and proven techniques).
- Establishment of a corporate repository of reusable software artifacts (realizing you can't reuse what does not exist, and therefore planning to develop appropriate reusable artifacts).
- Establishment of new rules and reward structures (promoting behavior that leads to the expected object-oriented benefits).
- Development of economic models to understand impact on costs, personnel allocation, and schedules (realizing object-oriented technology does not change the basic managerial responsibilities, but does alter aspects of existing models).
- Establishment of and adherence to a project schedule (reaching closure in the face of exploratory development).

Each month, this column will explore the above issues and other topics from the perspective of management and, wherever possible, the context of these explorations will be actual project experience.

Dr. Adele Goldberg *is President and CEO of ParcPlace Systems. Adele spent 14 years as a researcher and manager at Xerox Palo Alto Research Center prior to founding ParcPlace Systems. She has served as president of the ACM and has received the ACM Systems Software Award. She has writen extensively in the object-oriented field. She may be reach at ParcPlace Systems, Inc., 1550 Plymouth Street, Mountain View, CA 94043; (415) 691-6700.*

Languages and Environments

OOP in the 1990's

by Zack Urlocker

The beginning of a new decade is a good time to take stock of the events of the previous ten years. We can prepare for the future with the benefit of 20-20 hindsight, by putting significant events in proper perspective.

The year 1989 will be remembered by many as "the year of the object," because of the tremendous growth we've seen in object-oriented programming. There have been many significant events in object-oriented languages and environments, which signal an increasing acceptance of object-oriented programming, and that have significant impact on how OOP will be used in the future.

The OOP Market is Growing

According to Ovum Research's reports, the object-oriented programming market will grow from about $140 million in 1988 to about $2 billion by 1995. This estimate is based upon growth expected in the market for class libraries, program generators, and end-user tools. The industry has already anticipated the need for more information on OOP, and has created an infrastructure to deal with this need.

At the beginning of 1989, I toured the United States to speak with the editors of about a dozen leading computer magazines on industry topics. It was clear that OOP was the topic most requested by readers, and the magazines took this to heart. The magazines responded with varieties of articles ranging from tutorials, to design lessons, to product reviews. OOP appeared on the covers of no less than half the magazines reporting on it. OOP has also spawned its own specialized publications such as the *Journal of Object-Oriented Programming, The C++ Report* and, of course, the *Hotline on Object-Oriented Technology.*

The appearance of so many sources of OOP information indicates that the market is becoming strong enough for segmentation. I believe that the OOP marketplace could support at least half a dozen different publications devoted to the topic. Some segments of users are interested in research advances, some are interested in design methodologies, others are interested in practical hands-on techniques, and some are interested only in a particular language. I'm not predicting the launch of *OOPWeek,* though I do expect that over the next few years OOP will be covered regularly as part of every major software magazine, much the way that structured programming is covered today. In most cases, OOP information and techniques will become tied into the mainstream of programming forums.

OOP Usage Is Increasing

Interestingly, the OOP topics covered in books, magazines, and conferences are also changing. The "What is OOP?" tutorials that were prevalent through mid-1989 have been replaced by more sophisticated articles on reusability techniques, design, and examples of good OOP classes. Similarly, the books that are appearing now show increasing coverage of design and analysis, even though they may be devoted to a particular language, whether it be C++, Actor, or Object Pascal.

Questions to be Answered

- OMG still needs to bring IBM, Microsoft, and Digital into the fold in order to ensure that its work will establish universal standards.
- OMG has a broad charter that allows it to address all relevant aspects of object orientation That breadth, however, renders the organization devoid of the focus that may be required to make significant strides in any of the areas using object-oriented technologies.
- Because DBMS standards are not expected to be established until 1991 at the earliest, OMG's impact on the object-oriented database market is uncertain.

If you have questions about the OMG, contact:

Chris Stone, President
Richard Soley, Director
Object Management Group, Inc.
33 Lyman Street
Westboro, MA 01580
(508) 898-9690

If you have questions regarding The Persistent Object Storage Task Force, contact:

David Hentchel
Prime/CV
Prime Park
Mailstop 15-55
Natick, MA 01760
(508) 371-2182

Tom Atwood, *founder and chairman of Object Design, has pioneered the commercial development of object-oriented technology. Previously, he was one of the founders of Ontologic, which is among the first companies to introduce a commercial object-oriented DBMS. He has worked extensively in the field of database development at Data General and Digital Equipment Corporation.*

Project Management

Talking to Project Managers...

by Adele Goldberg

The use of object-oriented technology has an impact on the various levels of management in organizations, as well as on each member of the programming team. Other articles and columns in this newsletter focus on the way in which object-oriented technology changes the technical rules [sic] for each of the programming team participants. These programming team participants include application analyzers, system designers, code implementors and documentors, code testers, code maintainers, and system documentors. This column primarily addresses the manner in which the introduction of this emerging technology into an organization affects managers.

We begin with a short background to give perspective on where object-oriented technology came from, and on where it is going. The direction in which object-oriented technology is going today brings up many organizational issues. We draw attention to the concerns and expectations of management and technical participants at each level of an organization. Future columns will address many of these issues.[1]

Background Perspective

Object-oriented technology was created and has evolved primarily for the benefit of one team member—namely, the individual code implementor. The goal was simply to make it possible for an individual to create personally interesting, graphical applications, consisting of potentially complex relations among the software elements. For example, this individual might create a simulation of household finances, complete with scheduling and economic models. The individual user was expected to learn how to program, but programming was viewed as a personal and isolated activity.

Network computing, with its potential for sharing information in a dynamic, extensive fashion, was anticipated, but was not emphasized. Hence, most of the issues regard-

[1] Elizabeth Gibson, Training Manager for ParcPlace Systems, has been conducting interviews in order to study project management issues and experiences. If you are interested in sharing your project experience with her, please contact her at (415)691-6700 or gibson@ParcPlace.COM. Confidentiality will be honored; any portrayal in this column will, of course, be disguised if so requested.

ing the use of object-oriented technology that would naturally be considered by teams, large or small, were purposefully ignored during the early development years (notably at Xerox PARC, where the initial Smalltalk work was done). Neither were organizational and coordination issues developed at this early time.

Early development of OOT anticipated and extensively supported three activities for the individual programmer, who was not necessarily expected to be a computer scientist or software engineer:

- Management of the specification of complex information models.
- Customization of existing applications to meet personal preferences.
- Composition of innovative graphical user interfaces.

To realize the goal of empowering the individual, research efforts focused on creating a computer programming language that supported the description of events in the real-world (simulations), and on creating a set of techniques for graphical presentation and interaction. To support the use of the language and graphical techniques, a set of tools was also created. These tools combined conventional ideas for source code editing and debugging with navigational aids for locating and incorporating existing software components. The language and tools were augmented with a rich library of software components. The idea was to support the ability to create new software through composition and refinement of existing components, thereby letting programmers build from the work of (other) experts.

The resulting software development environment, made up of source code, source code editors and debuggers, code browsers, and inspectors, created a new programming style, namely, programming by refinement or exploratory programming. Programming by refinement refers to the ability to reuse existing software by specifying exceptions. Exploratory programming entails the incremental development of a program or a set of programs, based upon incomplete specifications. The programmer, uncertain of the complete functional and architectural details of the desired result, creates as much of the program as possible, and lets the interim result dictate the direction of further development. Interim results contribute to the development of the program by enabling early design review and feasibility analysis, as well as by allowing early testing by target users.

The fundamental technology used to create this new form of development environment was object-oriented technology. The environment manifested itself in the form of new languages, composed of objects as encapsulated representations of data and operations on that data, classes as descriptions of objects, and message-sending as the means to coordinate and to communicate among objects. The Smalltalk-80 language was the first system to consist entirely of objects, classes, and message-sending. Modifications were later made (and continue to be made) to existing languages such as Lisp, Pascal, and C, in order to add object-oriented constructs.

Graphical techniques evolved, popularly known as direct-manipulation interfaces and icon-oriented presentations. This happened as a result of the creation of visual representations of objects on bit-mapped display screens, which permitted both the programmer and the end user to interact with the representations. Dynamic cross referencing and navigational tools (browsers), and dynamic analysis tools (inspectors and debuggers), were easily devised. These tools came about because objects lend themselves to uniform query about their structures and interrelationships, the messages to which they respond, and the messages that they send to other objects.

Today's Perspective

By the late 1980s, the notion of object-oriented software became fairly familiar on the market. The benefits of exploratory programming and programming by refinement, and the advantages of the new tools, have gone to individual object-oriented application developers. These programmers typically create simulations of physical or conceptual process's dynamic models, such as financial or economic for example. The simulations have novel graphical interfaces for improving the users' understanding of, and involvement with, the underlying models. The benefits that attract the programmers are as follows: rapid application development, one-to-one mapping of descriptions (from real-world entities to interacting), computer-representable objects, and the opportunity to reuse the programmer's own previous designs and code.

The objected-oriented languages, tools, and techniques, which were created to empower the individual, could be rapidly and effectively adopted. This adoption could occur without a deep understanding of how this new form of development affects the more traditional software development lifecycle. This can happen in cases where the resultant software is delivered to a small population of users within the setting in which the software was developed, where the problems being solved are intimately understood by the programmers themselves, and where the solutions are often used by these same programmers.

The use of object-oriented technology, however, is now changing and broadening. As we enter the 1990s, object-oriented technology is being more widely deployed into the commercial sector. More specifically, traditional programming languages have added object-oriented constructs and libraries of reusable application components. Additionally, toolkits of reusable software objects for graphical user interface construction are available. Without commenting on the reason for it, the fact remains that the traditional programming community is now confronted with an insurmountable opportunity: to understand how best to manage the adoption of object-oriented technology in the form of languages, graphical techniques, tools, and libraries of reusable parts, all within the context of the traditional software development and usage lifecycle.

Organizational and Individual Concerns

The structure of organizations varies widely, but regardless of the structure, the introduction of object-oriented technology will have an impact at many levels In this hierarchy, we see project engineers, group heads, section heads, department and laboratory managers, and the division manager or vice-president, who is responsible for the collective whole. Of course, in a smaller organization, the layers of management might simply entail teams of product marketing and product engineering managers, who report to respective marketing and engineering directors.

For the department manager charged with obtaining or keeping a competitive advantage in the market, the introduction of object-oriented technology may be critical for keeping the business at the leading edge. This manager is often responsible for the full spectrum of activities pertaining to development, including contracts, research and development, business case formulation, and software development. Therefore, the department manager needs knowledge of the organization's competitive market situation and opportunities. Today, with rapid changes in the technology base, and with confusing stories about standards, it is notably hard to develop technologies and products with long-lived value-added aspects. Differentiation or innovation is required, but is difficult to sell in a world that is concerned with conforming. Introducing objected-oriented technology can create two benefits for the department manager: the ability to respond rapidly to changes in business assumptions, and the ability to respond to these changes in an incremental fashion that doesn't risk loss of standards.

Reporting to the department manager, the section head is charged with creating the desired competitive advantage. Where the department manager is responsible for multiple product sections, the individual section head is typically required to have more in-depth technical knowledge.

The section head monitors activity directions and allocates resources for each group, while the group head provides technical leadership for a particular product or research project. Both the section and group heads require competence in selecting the technological basis for a project, confidence in project planning, good control over project execution, and leverage from past projects. Especially troublesome in the introduction of a new technology is generally the control of the time to the completion of the task execution and milestones, as well as the ability to understand individual engineering contributions and productivity. These factors also affect, specifically, the exploratory development of and the implementation approach to object-oriented programming. The section and group heads may need to adapt their traditional project management tools and skills to work with the new object-oriented technology.

Project engineers perform the technical tasks. They require libraries of reusable components, tools, and other resources commensurate with meeting the expectations of their managers. Selection of technology and appropriate tools is a shared responsibility, although professional programmers prefer a flexible choice of complete tool sets, which support a smooth transition of information and function among the tools. Tools have to support each level of system implementation—tasks that are done by the individual (for example, fast edit-link-load cycle), and tasks that are done by groups (for example, multiprogrammer coordination). The introduction of object-oriented technology comes at a cost to the project engineer. The cost involves learning the development of a new programming language and applying new design techniques. New tools also must be devised and learned to support an incremental development style and to make good use of libraries of reusable objects. The payoff, of course, is in the ability to create rapidly and with greater confidence, innovative and complex applications.

End users are the customers of a software project. Today, endusers expect the applications that they use to be aesthetically pleasing (that is, with graphical user interfaces), easily (re)learned when used, infrequently integrated with other applications and system standards, open to being changed rapidly (not necessarily by themselves), and executable on any of the hardware platforms that are available to them. Their primary concern is that there is a significant time lag between the requests for changes and the updated results. Object-oriented technology allows developers to produce that which end users need.

Everyone's job is to create and to add value to the competitive advantage for the organization. The introduction of a new technology can be disruptive in the short run; we need to be aware of and to plan for this discipline, however, because the benefits are worth it.

About this Column

The issues that will dominate our studies of the impact that object-oriented technology has upon an organization are:

- Introduction of prototyping into the development process (budgeting for several stages of prototype creation, and getting value from the prototype).

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes,*** I want to be plugged into the latest thinking and developments in the object-oriented technology. Immediately enter me as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	2 years (24 issues)
☐ ~~299~~ $249	☐ ~~$598~~ $498

outside the U.S.—add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT—U.S.currency drawn on U.S. bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card #_____ Exp. date_____
Signature _____

Name _____
Company _____
Address _____
City _____ State_____ Zip_____
Phone_____

For immediate service—call the Hotline 212-972-7055 or Fax 212-370-4983

HOTLINE on
Object-Oriented Technology
The International Newsletter

310 Madison Avenue
Suite 503
New York, NY 10017

HOTLINE on
Object-Oriented Technology
The International Newsletter

March, 1990 **Volume 1, Number 5**

Objects of Desire

by Christopher M. Stone

When we talk about industry standards, we usually mean communication interfaces such as OSI, APPC, and so forth. Sometimes we mean programming interfaces such as Posix, Binary Compatibility Standards, and other application programming interfaces (APIs).

There is, in fact, another interface that needs to be standardized, perhaps even more than all the others. Until recently, however, the industry has paid little attention to this interface. This is the interface between the user and our information systems.

It is easy to forget that the investment of human resources in the industry is many hundreds of times the investment of computer hardware and software. One of the greatest challenges of the decade is to establish a Common User Environment, so that all users—managers and contributors, administrators, and engineers—can work readily in a unified and consistent environment.

A Common User Environment would give customers the freedom to deploy computing resources effectively throughout the enterprise. As the information system evolves, a Common User Environment will reduce the cost of retraining, and will minimize lost productivity. A Common User Environment would eliminate a major obstacle to truly open systems, that is, vendor independence.

Because the user interface is the most visible part of the system, it provides the most obvious opportunity for product differentiation. Vendors are in a difficult position: freedom of choice for the customer seems to be in conflict with marketing and business objectives. For this reason, there are vendors who claim to support a standard user interface, but only across their own product line. In light of this, the recent action by the Open Software Foundation (OSF) to select a User Environment Component, OSF/Motif, is a major victory for the customer.

To target this issue, and to take steps to resolve it in a mere six months, is an unprecedented event. By deciding to adopt the joint Hewlett-Packard/Microsoft/Digital proposal for OSF/Motif's look and feel, OSF has suggested implications that reach far beyond the UNIX community. Not only does the proposal provide a way to have a Common UNIX Environment, but it also will unify three of the four major workstation platforms of the next few years: DOS, OS/2, and UNIX (Macintosh being the fourth).

But standardization of the user environment should go much further than the "look and feel." Standardization should be based upon an object-oriented model, such as those offered by Hewlett-Packard's NewWave, NeXT Inc.'s NeXTStep, and others. Out of this concern was born the Object Management Group (OMG).

OMG

The OMG is a nonprofit, international organization of information system vendors

continued on page 3

Contents

Feature:
Objects of Desire
1

Thoughts From the Editor
2

Letter to the Editor
2

Databases
OMG: The State of the Object-Oriented DBMS Market
4

Project Management
Talking to Project Managers...
6

Conference Report
SCOOP-West and the Software Industrial Revolution
10

International Beat
11

Publicity Corner
12

Company Showcase
13

Company Showcase
13

Book Review
C++ Primer
14

Product News
15

In this month's *Hotline* the feature article is from Chris Stone of the Object Management Group, which describes the group's view of the requirements of standardization in the object-oriented world. Last month we began this discussion by focusing on the database requirements. This month's feature continues and expands the discussion.

Adele Goldberg (with Kenny Rubin) continues her column *Talking to Project Managers* with a discussion of prototyping. The column explores the use of OOT in prototyping and discusses the differences between using traditional prototyping tools and using OO tools.

In the database section, Tom Atwood discusses "OMG: The State of the Object-Oriented DBMS Market."

We have expanded the *Publicity Corner* as a result of the larger number of articles of interest that have appeared in the computer related press. You will also find more products and companies introduced in the *Product News* and *Company Showcase* sections.

We have here a review by Karen Cunningham of a good book on C++, and a brief review of SCOOP West.

All of these articles will stimulate you to think more about the growth and utility of object-oriented technology in other organizations, as well as in your own organization.

During the recent SCOOP West conference in Santa Clara, a reporter from the local paper interviewed a number of attendees and asked each "What is an object?" The reporter concluded from the answers that no one knew what an object was.

Richard Frank, the founder of Sorcim Corporation of spreadsheet fame, commented that people in general seem to be able to take simple concepts and make them complex.

I could not agree more. The very simplicity of object-orientation is a major stumbling block for many people. They believe that programming is difficult, and since one can program in objects, then objects must be difficult. My experience is exactly opposed to this view. Programming is only difficult if the machine is at the center of our attention. Once the application is at the center of our attention, and reuse is possible, then programming is as easy as our understanding of the application, and as good as the quality of our reusable components.

I have been observing a healthy scepticism in the press regarding objects. While many developers have been "converted," it is up to them to pass the word, primarily through the demonstration of complete, product quality software that has been developed using OO techniques. We are beginning to see those demonstrations now, and I anticipate that the number of new products using these techniques will increase dramatically over the near term.

At the same time, we need to see more theoretical work on objects so that we may better understand the strengths and weaknesses of objects. I would like to see less "work" that is just the development of new object-oriented languages but rather would like to see work that delves deep into the essence of objects, and that applies what is known about cohesion, coupling, information hiding, and other metrics, to objects. It has been remarkable to us that OO has proceeded as far and as successfully as it has with such little theoretical underpinnings. I would like to see a fresh look at objects, without the strong coloring of the limited past of computer science and software engineering.

These two activities, successful demonstration and strong theory, will do a great deal to bring object-oriented technology into the mainstream of computing.

HOTLINE on Object-Oriented Technology
The International Newsletter

Jeff McKenna
EDITOR

ADVISORY BOARD
Jim Anderson
Digitalk, Inc.
Tom Atwood
Object Design, Inc.
Grady Booch
Rational
Larry Constantine
Consultant
Cort DeVoe
IBM
Adele Goldberg
ParcPlace Systems
Bertrand Meyer
Interactive Software Engineering
Reed Phillips
Knowledge Systems Corp.
Trygve Reenskaug
Taskon A/S
Bjarne Stroustrup
AT&T Labs
Dave Thomas
Object Technology International
Zack Urlocker
The Whitewater Group
Eugene Wang
Borland International

Richard P. Friedman
Group Publisher
Xavier Yaffar
Technical Advisor
Melanie Younossi
Marketing Manager
Lori Auerback
Marketing Assistant

The Hotline on Object-Oriented Technology (ISSN#1044-4319) is published monthly by SIGS Publications Inc., 310 Madison Ave., Suite 503, New York, NY 10017. (212) 972-7055.
©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express persmission from the publisher.

Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30. Send address changes and subscription orders to HOOT, 310 Madison Avenue, Suite 503, NY, NY 10017
Submit editorial correspondance to Editor, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062

Letter to the Editor

Dear Sir,

We here at Project Technology were pleased to find PT mentioned in the article (*HOOT*, December 1989) on the OOPSLA '89 conference. However, I'd like to call your attention to an error: Project Technology's OOA method is currently supported by several software products. In addition to Cadre's Teamwork, which offers OOA support in the standard product, there are custom versions of InTech's Excelerator and Cadware's Sylva Foundry that support OOA. Adagen, from Mark V, has also announced OOA support quite recently.

Sally Shlaer
President,
Project Technology, Inc.

interaction style, an "on-demand" symbolic debugger, a "Visualizer" to visualize data during debugging, and many other tools to help the programmer debug and optimize programs. While programming systems for massively parallel systems are traditionally cumbersome and complex, the MP-1 family was designed to provide an experienced application programmer with the tools to take advantage of a graphically powerful system.

Designed to break traditional price/performance barriers of high-end computing, the MP-1 family will be used for computational chemistry, fluid dynamics, computer aided engineering, and image and signal processing. MasPar leveraged not only *Objectworks for Smalltalk-80,* but other proven technologies such as the DEC Ultrix™, operating system, AT&T Unix™ and VLSI chip design.

"It has been exciting to see the innovative people at MasPar take advantage of our object-oriented technology," said Adele Goldberg, President and CEO of ParcPlace Systems. "*Objectworks for Smalltalk-80* is gaining rapid acceptance for commercial systems and applications, as companies such as MasPar recognize how much they can accomplish by capitalizing on the work that has gone into it."

Objectworks for Smalltalk-80 provides software developers in corporations with integrated tools and reusable software components to create and deliver application. These applications can be as diverse as a customer service system for a financial institution, a manufacturing controls system for a large computer system company, and a programming system for FORTRAN and C, for MasPar's massively parallel system.

For more information, contact:
Helene Tannor
ParcPlace Systems
1550 Plymouth Street
Mountain View, CA 94043
(415) 691-6728

Borland's Reflex Wins Best Product of 1989

Reflex 2.0, Borland International's flat-file database manager, has been named one of the best products of 1989 by PC Magazine.

Reflex uses Borland's VROOMM™ (Virtual Run-time Object-Oriented Memory Manager) technology to provide more features in less memory. In addition, Reflex 2.0 offers a number of features that are not available in competing products, such as Symantec's Q&A and Professional File from Software Publishing. These features include multiple views, hot-linked windows, interactive crosstabs, and built-in, professional-quality graphics.

"Reflex 2.0 is packed with all the features anyone could possibly want from a flat-file database system," PC Magazine said. It runs on basic configurations from a 286 machine to an 8088-bases PC, with 640K of memory, a graphics card, and DOS 2.0 or later.

"We have redefined the flat-file database category by incorporating object-oriented technology to use computer memory more efficiently and provide more features," said Philippe Kahn, chairman, president, and CEO of Borland. "Reflex 2.0 can easily handle tasks other flat-file managers can't tackle."

Users need not understand the technology behind Reflex 2.0, according to PC Magazine, but can just "sit down and put Reflex 2.0's powerful resources to work on the problem at hand."

For more information, contact:
Rebecca Furtado, (408) 439-1623
Marguerite Padovani, (408) 439-1622
Borland International, Inc.
1800 Green Hills Road
P.O. Box 660001
Scotts Valley, CA 95066-0001

IBM Makes Equity Investment in Rational

IBM Corporation and Rational announced that IBM has acquired an equity interest in Rational. Rational is a leading provider of computer-aided software engineering products and technology, used to design, build, and maintain complex software applications written in the Ada programming language. Rational is headquartered in Santa Clara, California.

Under the terms of the equity agreement, IBM has made an investment in Rational in the form of newly issued voting convertible preferred stock. Additional details of the equity agreement were not disclosed.

The two companies also announced marketing and development agreements that will focus on providing a high-function, comprehensive Ada solution for customers in the federal and aerospace marketplaces. Ada is a software language that originated with the U.S. Department of Defense and it has evolved for use throughout the information processing industry.

Rational provides an integrated, interactive software development environment, based upon and supporting modern software engineering practices. This highly automated system supports the major tasks within the development life cycle, such as requirements analysis, design, and code generation.

Rational's products run on its own system and, in conjunction with IBM Ada compilers, produce software applications that run on IBM System/370 platforms and the IBM RT PC. Rational also provides integrated capabilities to develop and debug Ada applications for embedded computers.

IBM and Rational intend to enhance Rational's Ada products to run with IBM's Advanced Interactive Executive (AIX)* reduced instruction set computer platforms.

IBM has been using Rational's software engineering tools and technical consulting services on a number of Ada-based complex systems projects that it is handling for federal and aerospace customers. One project example is the Advanced Automation System (AAS), a program for modernizing the nation's Air Traffic Control System, which IBM will develop, deploy, and service.

For more information, contact:
Nellie Connors
Rational
(408) 496-3692
Judy Radlinsky
IBM Corp.
(914) 642-4634

continued from page 1

with diverse backgrounds, including PC's, Mainframes, networking, and software that have united for the first time in the spirit of cooperation, in order to promote the concept of object management. Presently, 52 companies are OMG members—AT&T, Data General, Hewlett-Packard, Sun Microsystems, Nixdorf, Prime, Netwise, and Unisys, to name a few. In April of 1989, in New York City, the OMG committed itself to supporting the OMF (object management facility) of HP's NewWave. This included a commitment to work towards object distribution, databases, and system services.

Standards in all but a small number of cases are set either by standards-prescribing organizations, or by universal acclamation. Companies with more than 60 percent market share can effectively define standards by virtue of their customer base being, by default, in compliance.

Defacto standards require a number of necessary ingredients. The company attempting to set the standard must have credibility, or it must be first in the market in which it intends to have a standard accepted. The company derives these ingredients from technical excellence, recognized ability to deliver, and an idea that has broad appeal. The technology being proposed must be optimal. The technology must be a complete and efficient solution, and must be flexible enough to be extended to meet future needs. In addition, the technology must be readily and easily available. Finally, the company attempting to set the standard must publicly state its intention to create a standard, so that potential adopters will be confident that they will receive appropriate support, should they decide to implement.

To accomplish the goal of successfully implementing standards, OMG is promoting the adoption of object technology and specifications in five areas: a reference model with defined terminology; a set of APIs for use across multiple operating environments and user interfaces; distributed object management; interfaces to object and existing DBMSs; and common services, such as security and system management.

Converging on One Standard

The OMG believes that by getting the industry to converge on one standard for object management, vendors will be able to deliver quality software products more quickly. In addition, programmers will be writing for one standard, and they won't have to develop different versions of their product for multiple environments.

Standards are useful because they make two promises: if a product is built that is compliant with standards, it will be interchangeable with, and will interoperate with, other products complying with the standard; also, if a product is compliant with standards, it will be a known quantity in the marketplace.

Standards raise the customer's comfort level. Products complying with standards will result in a better quality product, shorter debug cycles, lower long term maintenance costs, and the ability to react to changing market conditions. Designing to a standard, such as the OMG's guidelines, will obligate the developer to build the product correctly, and to abandon the route of short-term expediency.

Today's software systems force users into a procedure-data model, even though they are probably much more comfortable with an object model. Our systems do this, not because programmers get perverse joy from it, but more or less because it is a byproduct of our programming languages. Standardized object-oriented technology will take us away form this procedure-data model.

Our information systems can be seen as six layers. Hardware is at the bottom. On top of hardware are systems and networks, a data management layer, an applications layer, and a user interface layer. At the top of the stack is the user (this is not an architectural model but a conceptual one). It is essential that the user be included in the model; in today's systems, the user is not just a passive beneficiary of the technology, but is very much an active part of the system.

The active part that the user plays is really the crux of the problem with conventional systems design. Our systems aren't really smart; they are just dumb and fast. The user still bears most of the responsibility for being smart. As we make our systems of the 1990s more object-oriented, we will be forcing "smarts" into the system, and we will be taking the burden from the user.

There are numerous examples of object-oriented systems, implemented at the workstation and user interface layers, under a variety of "standards." The Macintosh is a good example of an object-oriented interface. NeXTStep is a fairly thoroughly object-oriented development environment, with visual programming flair. But all of these examples stop at the network. Network access is still governed by the old model.

The user must have access to network services and applications. To be useful, these need to be visible. The user does not need to be concerned about, or even to see, where the services are physically located and how they are implemented. This can't happen unless a group like the OMG sets standards in front of implementation.

Therefore, while the workstations of today are connected to conventional networks, the workstations of tomorrow will be windows into a standard, object-oriented network of services and resources.

There are two, real world requirements for standard setting in the object-oriented arena. The first is that the technology must be portable. For an object-oriented system to be practical, it must run in a wide variety of environments. The system should run on any "industry standard" implementation of UNIX; it should not be bound to a particular communications stack. The system should also be capable of working in a client/server pair with any number of workstations.

The second requirement is that OMG standards should be flexible enough to interoperate with with systems that now exist. The standards should be able to take advantage of any resource that is already part of the underlying operating system, and they should be able to communicate with existing application systems with few modifications. They should be able to act as a front-end to any industry standard relational DBMS engine or SQL.

In other words, any object management system should be able to be integrated with with the conventional application systems of today—while we are developing the the fully object-oriented distributed systems of tomorrow.

But How?

Creating standards, before the ubiquitous word "defacto" creeps into the vernacular, is the primary mission of the OMG. Making this a reality is, in some ways, easier than it sounds. The group feels that it is infinitely easier and cleaner to define where the concept is going, and then to begin the task of getting there, than it is to look at where many manufacturers, working to their own philosophical and economic ends, have taken the concept thus far. Creating the standard after the fact, although the custom in this business, is the opposite of the OMG's goal.

We believe that we have 12 months to get our house in order, and to begin to deliver. When defacto standards gain a foothold, marketing takes precedence over technology, and user-centered computing suffers. Instituting an agreed upon structure early, as the OMG is busy doing, will foster creativity and will direct the efforts of many people toward a common goal.

We have found that the best way to institute this structure is to uncover the object-oriented research and development that is now ongoing, and to recognize it as useful and innovative. We then harness the power, and direct it towards standardization across many software environments. When this happens, we will begin to have open software, just as we now have open hardware. Keeping our members and their products in front of the technology curve is another way in which we will keep people on the same wavelength. When the object-oriented explosion really happens, we will have built a convention without preconceived methodology or political motivation.

Christopher M. Stone *is a founder and president of the Object Management Group (OMG). Mr. Stone was formerly group manager and director of software at Data General Corporation. He may be reached at: OMG, P.O. Box 395, Westboro, MA 01580, Telephone: (508) 898-9690.*

OMG: The State of the Object-Oriented DBMS Market

by Tom Atwood

Between now and OOPSLA '90, in October, the object-oriented DBMS market will undergo more change and more excitement than at any time in its history. New products will be announced and shipped, which will represent major advancements in the state-of-the-art; standards discussions will become increasingly active, and the industry will enter a phase of rapid growth.

Now seems an appropriate time to discuss the state of the object-oriented DBMS market. *The Object-Oriented Database System Manifesto* and the Object Management Group's plans and programs have both been presented recently (December and January *HOOT*), and as promised, I will comment on the implications of these plans and programs for the industry. I will then address two questions that I am asked frequently. Finally, I will make some predictions about what will happen in the future.

Comparison of the Manifesto and OMG

The goal for the OMG and the Manifesto authors is to foster the development of object-oriented technologies. While the Manifesto concerns itself solely with object-oriented database systems and languages, the OMG is trying to set industry standards for a complete object-oriented development environment, of which database systems are a part. In effect, the OMG and the authors of the Manifesto are working to define the requirements that a database must satisfy in order to be considered an OODBMS.

Both the OMG and the Manifesto authors also recognize that trying to establish standards too early will stifle the innovation and creativity necessary for object-oriented database systems to succeed. This recognition is to be expected from the Manifesto's authors, since they are the leading academicians in the field, and research is crucial to them.

Attempting not to establish standards too early could pose a real problem for the OMG, since its mission is to foster standards. Fortunately, the OMG recognizes that OODBMS technology will advance dramatically in the next year, and the group has established a course that will not attempt to set database standards prematurely. Since all the OO-DBMS companies are OMG mem-

prehensive environment for developing high-quality software.

Single license prices range from $5,000 to $21,000, depending on the selected configuration.

For more information, contact:
Nobby Akiha
Interactive Development Environments
595 Market Street, 10th Floor
San Francisco, CA 94105
voice: (800) 888-IDE1
fax: (415) 543-0145

Eiffel Institute of Technology

Interactive Software Engineering, Inc. announced the opening of the Eiffel Institute of Technology (EIT) in early March of this year. The Institute will be in Santa Barbara, California. The aim of the Institute is to help software engineers and managers to understand the modern concepts and techniques of object-oriented design and programming, and to provide product training in Eiffel, an advanced object-oriented environment developed by Interactive.

The software industry's growing interest in object-oriented techniques has raised a considerable need for quality training in this field. The Eiffel Institute of Technology is a direct response to this need. The Institute expands Interactive's program of seminars on object-oriented design and programming, which have been offered to thousands of attendees worldwide. A related effort is the Eiffel University Partnership, already in place, which targets accredited institutions.

According to Bertrand Meyer, President of Interactive, "The rate at which Object-Oriented Programming in general, and Eiffel in particular, is being accepted by the industry is staggering. We realize that what the industry wants is a full curriculum of courses on object-oriented programming and other advanced software engineering techniques. The EIT will be the first full-scale institution entirely devoted to teaching this technology."

Describing the EIT's curriculum, Tom McCarthy, Interactive's Vice-President of Product Marketing, said "Eiffel and object-oriented programming are expanding in depth and breadth; our curriculum must reflect this. Courses will be offered along four tracts: object-oriented design, object-oriented programming, project management, and general topics." The 1990 curriculum includes eleven courses, offered with various frequencies.

The course titles include:

- Object-Oriented Technology: A Management Overview
- Object-Oriented Software Construction
- Building Professional Libraries of Reusable Software Components
- Advanced System Design
- Basics of Eiffel Programming
- Intermediate Eiffel Programming (3 courses)
- Building Graphical Applications
- Managing Large Object-Oriented Projects

For more information, contact:
Darcy Harrison
Interactive Software Engineering Inc.
270 Storke Road, Suite 7
Goleta, CA 93117
voice: (805) 685-1006
fax: (805) 685-6869

SIMULA

The general purpose programming language SIMULA was developed in Oslo at the end of the 1960s as a language for simulating the behavior of systems of discrete events. SIMULA has been the forerunner, direct or indirect, of all other object-oriented languages, such as Smalltalk and C++. SIMULA is now a powerful, up-to-date, general purpose language, which is also suitable for simulation tasks.

SIMULA is available on all major mainframes, minis, and micros. PC SIMULA is available in DOS, XENIZ, OS/2, and UNIX/386 operating systems.

Prices vary depending on configuration, and are available upon request. SIMULA is available immediately.

SIMULA is used by major universities all over the world, as well as by government agencies and private industry. Since SIMULA is such a popular tool for teaching, research and development at universities, Simula a.s. is now offering a discount of 40% on all of its SIMULA System prices for universities, technical colleges, and students.

For more information contact:
Simula a.s.
Postboks 4403—Torshov
N-0402 OSLO 4
Norway
voice: (+472) 15 67 10
fax: (+472) 15 60 51

Objectworks for Smalltalk-80 from ParcPlace Systems Used as Basis for Programming System for New MasPar Computers

ParcPlace Systems and MasPar Computer Corporation jointly announced that *Objectworks for Smalltalk-80* was used to build an advanced programming environment for MasPar's new MP-1 family of data parallel computers. The MP-1 is a new class of high performance systems, based on massive parallelism and an advanced programming environment for FORTRAN and C programmers.

Objectworks for Smalltalk-80 is a graphical, interactive, object-oriented development system that facilitates the design and use of reusable software. The product provides a seamless interface to the computer platform, as well as true application portability to other hardware and software platforms. The system's rich set of libraries, object-oriented language, and tools enabled MasPar to create a specialized environment for the FORTRAN and C programmers.

"Using ParcPlace's Objectworks for Smalltalk-80 offered a unique opportunity to enter the market quickly with a breakthrough in supercomputer programmability" said Peter Christy, MasPar's Vice President of Software Engineering. "Objectworks provided both an elegant and mature tool kit for developing the kind of user interface we needed to accomplish our product goals, and the high-productivity tools a start-up needs to meet short product development cycles."

The MasPar software development environment includes a "point and select"

"We are delighted that ParcPlace is porting its advanced object-oriented C++ development system to RISC platforms like SPARC™," stated Mike DeFazio, Director of System Software at UNIX Software Operation at AT&T. "The UNIX Software Operation is committed to supporting C++ and encourages its use among developers. We believe that C++ can provide significant productivity gains in programming graphical and high-performance applications as well as system software."

The integrated set of object-oriented development tools addresses key phases of C++ application development from original design through reuse and maintenance. *Objectworks for C++* includes five essential components: a source code browser with multiple code views; as incremental C++ compiler including AT&T's C++ Language System 2.0; an incremental linker; a source level process inspector for debugging code; and a mouse-oriented editor.

President and CEO of ParcPlace Systems, Adele Goldberg, observed that "with the availability of *Objectworks for C++* on the SPARCstation family, ParcPlace is demonstrating its commitment to the growing C++ community by providing object-oriented programming technologies to C++ programmers on a variety of computer systems. Moreover, we are keeping pace with Sun's development of new workstation technologies."

Objectworks for C++ will be available on the SPARCstation family during the second quarter of 1990. Pricing is set at $2,495 for both the Sun-3 and SPARCstation platforms. Software support is available, including a customer hotline, upgrades to new software releases, an electronic bulletin board, and newsletter.

For more information, contact:
Helene Tannor
ParcPlace Systems
1550 Plymouth Street
Mountain View, CA 94043
(415) 691-6728

"Action!—Oriented Programming™" on the Macintosh

ExperTelligence, Inc. announced the first integrated symbolic software application development system, available for the Macintosh II series. The Action! Developer System for the Macintosh II series is the first CASE tool available for the Mac that integrates the Action!® technology, originally developed by ExperTelligence on the microExplorer™.

The development system includes the Action! interactive application development tool, developed by ExperTelligence, the Macintosh Tool Box Interface (TBI), developed by ExperTelligence and adapted by Procyon Ltd., and the Procyon™ Developer system, developed by Procyon Ltd. and distributed by ExperTelligence.

With this system, a developer can develop an application on a native Macintosh, compile it, and deliver it standalone in record time and with unprecedented quality. Applications that could have taken from three to five man-years can be developed routinely in less than six man-months. Not only are they much easier to use and more attractive, but they are also bug-free from day one!

Denison Bollay, President and Chief Executive Officer of ExperTelligence, said: "This system is the most advanced symbolic development system available on any machine—except for the microExplorer—today. Even traditional Lisp machines or standard workstations cannot offer such a powerful set of integrated tools. This is also the only development system available for any microcomputer that takes full advantage of the power of the international CLOS (Common Lisp Object System) standard."

The development system can work on any Macintosh II system with a minimum of 5 megabytes (8 megabytes recommended). Applications generated by the system can run on a 5 megabyte configuration. The full development system is available now from ExperTelligence, at the introductory price of $3,000. For this price the customer receives six binders of high quality documentation, 25 diskettes containing the various programs, hundreds of example programs, a free subscription to the "Action! Tech Notes," a free subscription to the "Inspector" and the "ExperNewsletter", free HotLine support, and free access to the ExperLine™, which is an electronic proprietary technical support network. Additional training and consulting are available upon demand. A complete seminar program will be announced during the first quarter of 1990.

Dr. John E. Forge, Executive Vice President and Chief Operating Officer said: "With this product, we are answering a growing need of the corporate developer market. After the delays and problems encountered by most of the large software developers to bring their products to the new graphical interface systems now available on microcomputers, more and more corporations are starting to understand that Object-Oriented programming is the only solution, and symbolic programming the best alternative. We can now provide these corporations and government organizations with the best tools and the most complete service available from any microcomputer developer."

For more information, contact:
ExperTelligence, Inc.
Bernadette Bagley,
Director of Communications
Dr. John E. Forge,
Executive Vice President
5638 Hollister Ave., 3rd Floor
Goleta, CA 93117
voice: (805) 967-1797
fax: (805) 964-8448

Software through Pictures ® Available for DEC ULTRIX™ with DECwindows ™

Interactive Development Environments, Inc. (IDE) announced the availability of Software through Pictures ® Release 4.2 for all of Digital's ULTRIX™ workstations, including the DECstation™ and VAXstation™ families.

Software through Pictures is an integrated, multiuser CASE environment. Built upon an open architecture, Software through Pictures can be extended, customized, and integrated with other tools to meet users' needs. This new release on ULTRIX uses DECwindows™, Digital's implementation of the X Window System™, and can be used to implement an X11-based Software Development Center. Release 4.2 is a major new release of Software through Pictures on ULTRIX platforms, providing several new user-extensible applications that are built on top of template-driven tools. IDE supports DEC's CASE strategy of providing a common

bers, the OMG will be an excellent forum for formulating standards. And it is clear that OMG will be an important group to watch in the year ahead.

Questions I am Frequently Asked

In the course of my activities I meet many people who are interested in OODBMS and the way in which they compare to relational database systems. Two questions are raised consistently.

If Codd's rules are not followed, how do we know that database integrity is maintained?

Codd's rules are valid only for relational database systems, and are only necessary because of the tabular implementation of the relational data model. Just as Codasyl and network databases guarantee data integrity without Codd's rules, object-oriented database systems ensure data integrity. In fact, the inherently richer semantics of the object-oriented data model make it easier to provide for data integrity. For example, in an object-oriented data model, bidirectional relationships can be automatically maintained, thus guaranteeing referential integrity. Data is encapsulated and can only be manipulated by methods (operations). And methods can be coded to do any number of useful verifications when invoked. Furthermore, object-oriented languages, such as C++, are strongly typed. In sum, the requirement of ensuring data integrity is met.

I hear is a lot about the productivity advantages of object-oriented programming. Where is the supporting data?

Two points are relevant. First, object-oriented programming encourages module and code reuse, and thus the old metric of measuring programmer productivity by counting lines of code produced, is not valid. A case in point is our experience at Object Design with coding in C++. While it is impossible to measure accurately, we believe that the functionality provided in ObjectStore has been achieved in about *half* the number of lines of code that it would have taken in C. We have also achieved other benefits. First, we feel that code modularization is superior. This has made testing easier and will make ObjectStore easier to maintain and enhance. Finally, on a code per programmer basis, we have achieved a high number versus industry standards. Despite the fact that we have an outstanding technical team, more is at work here.

Perhaps the best way to gauge productivity is to look at AT&T's results. In our inaugural article, we noted that after two to three years of use, AT&T is now enjoying significant benefits from code reuse. Dave Belanger, head of the Advanced Software Department at AT&T Bell Laboratories, reports that they have measured a large number of software efforts, and that "a C application will have three times the lines of code as a C++ application offering comparable levels of functionality."

I know that more and more people are using C++, and if you have stories that you can share, please call or write me at the address below.

Predictions

Finally, let me share with you my projections for major events in OODBMS in the next year. As I mentioned, new products will be announced and shipped that represent major advancements in the state-of-the-art; standards discussions will become increasingly active, and the industry will enter a phase of rapid growth. More specifically I predict that in the next twelve months:

1. Each of the next generation companies, Object Design, Objectivity, and Object Sciences will announce products before OOPSLA '90. They will offer true DBMS functionality.
2. CAD/CAM CAE firms will begin serious use of third party OODBMSs.
3. Object-oriented databases will find applications beyond CAD/CAM CAE, including advance office automation, financial applications, and multimedia applications.
4. OS/2 will become an important focus of activity for OODBMSs.
5. C++ will continue to set the standard for object-oriented languages in the workstation and PC world. AT&T will announce plans for C++ 3.0, and will ship within a year.
6. We will see increased discussion of OQL, an object query language that is a true data manipulation language (DML) for the object generation. This will be analogous to SQL in the relational world.
7. We will see the OODBMS companies announce plans for:
   —porting to all major platforms
   —more front end tools including report writers, forms managers
   —integration with existing DBMSs.

For those of you who are trying to find the Manifesto, it was published in the Proceedings from the Deductive Object-Oriented Database Conference held in Kyoto in December, 1989. Alternatively, if you would like a copy please write to me : Tom Atwood, Object Design, Inc., One New England Executive Park, Burlington, MA 01803. Tel. 617-270-9797

Talking to Project Managers...

by Adele Goldberg & Kenny Rubin

The "software crisis" has many forms, depending on where one sits in an organization. Managers of software development are confronted with a myriad of problems, including:

- Tendency to deliver products that do not meet end user needs, and that are not easily employed by those end users
- Consistent cost overruns and inability to deliver on time
- Inability to respond quickly and effectively to the inevitable requests for changes in requirements and functionality
- Too many projects waiting to be done, which tend to push innovation to the "backburner"
- Insufficient number of skilled programmers available

Despite requirements statements, surveys, and other instruments for soliciting end user input, users are rarely able to express clearly what they want, and users often prefer something other than what they get. Unfortunately, this is the most costly type of development problem, because, frequently occurs very early in the project (during specification), and often is detected very late in the project (at delivery). This situation results, in part, from the fact that the decision makers lack key information needed at each stage of development. Prototyping provides a mechanism by which the project may be better understood, and hence it may provide early, valuable input to the managerial decision-making process. In addition, prototyping is a useful technique for increasing user involvement and confidence.

Because of the many projects that are on the request list, members of the typical software team today have to decrease the numbers and kinds of risks that they can take to be innovative. At the same time they increase the number of new and different computer software technologies with which they deal. For example, in order to create even the simplest applications, programmers need to know how to conform to standards. They need to know about the aesthetics of graphical user interfaces, and details about open systems technology. Although developers are trying to understand user needs, the increased complexity of systems and the decreased ability of users to account for that complexity in their requirements statements, often leads to user disappointments.

Prototyping is a Proposed Solution to the Problem of Balancing the Increasing Demands on Programmers with the Decreasing Ability to Articulate User Requirements.

The purpose of this article is to examine how prototyping can help solve some software development problems.

What is a Prototype?

A prototype is an intentionally incomplete or scaled-down version of a system that contributes to the process of building the full-scale system. It is not sufficiently engineered for robust product delivery, yet the existence of the demonstration gives evidence that the full product can be developed. The term "rapid prototyping" refers to the process of quickly building and evaluating a series of prototypes.

Many disciplines employ prototyping as a standard technique. For example, an architect constructs a physically scaled-down prototype of a house, skyscraper, or landscape, placing the prototype in a context resembling the actual environment (a street, a set of other houses or skyscrapers, and so on). The architect does this in order to provide a visualization of the proposed result. An automobile designer creates a hand-built full-scale working version of a new car, which can be analyzed to determine a cost-effective approach to manufacturing.

Both of these examples illustrate important reasons for building a prototype. First, the prototype serves as a proof-of-concept. It is demonstration that the product concept is feasible and implementable within a set of constraints. Second, the prototype is a vehicle by which users may provide feedback on the completeness of the specification. A software development prototype exhibits these characteristics by providing an executable model of selected aspects of the proposed system. Because the software prototype is (partially) functional, it can be explored by the target users in order to determine whether the functional specification is complete. For example, a prototype of a system to support a manufacturing process, written using ParcPlace's Objectworks for Smalltalk-80, helped the designers and users to understand the extent to which their manufacturing methodology was incomplete.

Connell and Shafer [2] describe a prototype as a dynamic visual model that serves as a communication tool between end user and designer. Their book is written from the fourth-generation language (4GL) perspective of display-screen painting, where users can create mock-ups of screens for retrieving data from a relational database. Much of their discussion is appropriate when considering prototyping with object-oriented technology as well. Specifically, their discussion is useful in describing what prototyping is, when it might be useful, and how an organization would adapt to using it. The features common to 4GL and object-oriented rapid prototyping indicate that a prototype is:

- A requirements definition medium
- A means for providing users with a physical representation of key parts of the system before full implementation
- Functional after a modest amount of effort
- Flexible, so that modifications can be done with minimal effort
- Not necessarily complete, nor intended to be the final system

The differences between 4GL prototyping and object-oriented prototyping are primarily the specific techniques that are available for obtaining a prototype, and the expectations of what can be prototyped. According to Connell and Shafer, prototyping is a declarative process for:

- Creating a working model using a 4GL (during prototyping, no procedural (3GL) code is written)

Product Announcements should be sent to the Editor, HOOT, 8825 SW Umatilla St., Tualatin, OR 97062.

Object-oriented System Development Using ObjectOry is Spreading

Recently, Objective Systems has signed two important licence agreements. Ericsson Telecom will use ObjectOry in a development project for telecommunications equipment. The agreement makes ObjectOry available for all companies within the Ericsson Group, after evaluation of the current project.

Ericsson is one of the world's leading suppliers of systems and equipments for telecommunication. It is known for advanced technology and solid system expertise. The AXE telephone exchange system is one of Ericsson's well known commercial successes. The Ericsson Group has more than 65,000 employees, and is operating in about 80 countries.

The other ObjectOry licence is the Swedish Defence Material Administration. They will use ObjectOry for the development of both information systems and embedded systems. The agreement also makes ObjectOry available to the leading companies in the Swedish defense industry.

Objective Systems has also signed a cooperation agreement with the Swedish consulting company Frontec. The purpose of the agreement is to market system development help, where Objective Systems supports the development method ObjectOry, and Frontec provides trained consultants. The agreement will significantly increase the number of developers proficient in ObjectOry, which will lead to increased use and awareness of object-oriented development.

ObjectOry is an industrial approach to development and support of large software systems. It has been used by several large companies in a number of development projects, covering both technical and business types of applications. Some examples are system software for LAN management, spare part administration, and mobile communications. An interesting application is reverse engineering, by recreating an analysis model of an existing system. The choice of ObjectOry is a strategic decision. It makes available a technique that leads to a competitive advantage.

The complete system development process, ObjectOry, is offered as a package consisting of manuals, CASE support, education, and consultancy. It is developed and marketed by Objective Systems, Kista, Sweden. These new orders are a significant step forward for Objective Systems and the acceptance of ObjectOry. The company has, in a few years, become known as a world leader in object-oriented system development.

For more information, contact:
Lars Wiktorin
Objective Systems
Box 1128
S-164 22 Kista
Sweden
voice: +46-08-703 45 39

Peritus International Announces Release of its C/C ++ Compiler for UNIX 386/486 Systems

Peritus International announced the release of its ANSI C/C++ compiler for Intel 386/486 UNIX systems. The compiler serves professional software developers building large C-based system or application programs.

The compiler is one of the first highly optimizing C compilers to support fully all three major "C" dialects within one package, and to provide full translation support for object-oriented C++ development.

The compiler supports the popular K & R and ANSI dialects of the C language, as well as the newly emerging AT&T C++ standard. Programmers can select the C dialect of their choice simply by setting a switch in the compiler. This allows programmers to move smoothly between or from C to C++ without having fully to redevelop existing code.

Peritus C++ is a "true" compiler, providing compile speeds as much as twice the speed of pre-processor based C++ tools. The requirement for efficiency in a C++ compiler is especially pronounced because of the use of C++ class libraries—which are predefined modules of C code called by C++ programs, adding significant source lines to an application.

Based on Peritus's unique supercomputer compiler technology, Peritus C++ also provides a migration path for users who wish to develop programs on an i386/486 system and upload it to a mainframe system, or vice versa. The compiler supports an extensive set of data types which make this migration possible. These data types, include 8, 16, 32, and 64, bit integers, IEEE compatible 32, 64, and 80 bit floating point, user defined aggregate types, and C++ class data types.

Mainframe quality optimizations have been incorporated in the compiler, providing substantial code speed and flexibility options beyond those offered by existing i386/486 based C compilers. These optimizations include: global register allocation, constant propagation and folding, backward code motion with loop invariant removal, induction variable elimination, redundant store and dead code removal and constant elevation. Benchmark results show a 30 to 40 percent performance advantage over the standard UNIX "C" compiler.

The Peritus C++ compiler is currently available for i386/486 systems under SVR3 UNIX and SunOS 4.0 UNIX. The compiler sells for $1,000 on these systems.

For more information, contact:
Ron Price
Peritus International
10201 Torre Ave., Suite 295
Cupertino, CA 95014
voice: (408) 725-0882

Objectworks for C++ for Sun SPARCstation Family

ParcPlace® Systems today announced *Objectworks™ for C++* on the Sun SPARCstation™ family. *Objectworks for C++* is the first fully integrated development system that allows the programmer to exploit the object-oriented features of C++. *Objectworks for C++*, initially announced in June 1989 on the Sun-3™, integrates the AT&T C++ Language System, Release 2.0, with a set of tools designed specifically for the C++ language development.

With this new release ParcPlace's *Objectworks for C++* is now available on both CISC- and RISC-based platforms.

C++ Primer

(Addison-Wesley, 1989)

by Stanley B. Lippman

Most books on C++ treat the language as an extension of C. They typically begin by showing the small enhancements C++ makes to C and then they discuss the class construct in detail. While this treatment does enable existing C programmers to learn C++ (and most new users are C programmers), the treatment portrays the language as being a loosely connected set of features hung onto C. The *C++ Primer* by Stanley Lippman, is a notable exception to this norm.

Lippman treats C++ as a cohesive whole, and in so doing not only makes the language more attractive to prospective users, but also treats the features more thoroughly than other texts. As part of the AT&T Release 2.0 team, Lippman is well-versed in the newest features of the language. This expertise is also clear from the book; the book is by far the best treatment of release 2.0 currently on the market.

The book starts in Chapter 0 with a discussion of the compilation process and the preprocessor, topics that are frequently omitted from other texts. The next three chapters cover data types, expressions and statements, and functions. The class construct is discussed in Chapter 1, albeit rather superficially. I find it refreshing to have classes treated as an integral part of the type system and not as a mystical and difficult addition later in a book.

In Chapter 4 Lippman discusses storage allocation and overloaded function calls. This chapter represents a real strength of the book. Lippman's discussion of the argument matching process is the most complete and comprehensible I've ever read.

The next two chapters deal with the class construct. Lippman thoroughly discusses all aspects of using a class, including information hiding, constructors, operator overloading, assignment and initialization, and type conversion. The examples are real enough not to be boring, and simple enough not to be overwhelming. He provides reasonable examples of overloading a variety of operators, including an interesting use of overloading the operator () (function call).

The book sounds pretty good so far, doesn't it? Is this the final word on C++ books? Unfortunately not. *C++ Primer* is an excellent language text, but it falls short on the why and how of object-oriented programming. The book I'd really like to see written would include in-depth practical examples, using object-oriented programming to its best advantage. That book does not yet exist.

Chapters 7 and 8 cover object-oriented programming, and are a disappointment. There are a small number of examples of inheritance, and even fewer of polymorphism. As in most C++ books, the multiple inheritance examples are the weakest in the book. The only example of multiple inheritance that appears is about zoo animals, something I never have programmed, and most likely never will program. This type of example makes it difficult to justify the use of multiple inheritance to novices. Advanced features, such as multiple inheritance are useful to solve hard problems and it can be difficult to present such a problem in a small amount of space. There are problems that are approachable and interesting where these advanced features shine. An example, which I frequently use in C++ courses, is a screen-oriented text editor.

The lack of object-oriented features is a definite weakness of the book. I would recommend that anyone who wishes to understand object-oriented programming should consult some additional references. Fortunately Lippman provides a good list of references in his preface. Some of my favorites from his list are Johnson and Foote's *Designing Reusable Classes* and Koenig's *An Example of Dynamic Binding in C++.*

The book includes four appendices, all of which are very useful. The first appendix is a complete introduction to the C++ I/O library. In addition to the usual treatment of overloading << and >>, Lippman discusses several advanced topics, such as file input and output. The second appendix discusses future implementation of generics (parameterized types, classes, and functions) in C++. Release 2.0 C++ is contrasted with C and with release 1.2 C++ in the remaining two appendices. Programmers moving either from C, or from release 1.2 C++, to release 2.0 C++, will find a great deal of useful information in these appendices.

I have recommended this book to novices and to experienced people in the six months it has been on the market. It is encouraging that both of these groups of people have had positive things to say about the book. I feel it is a good book from which a wide variety of people can learn. Experienced C programmers should not be put off by coverage of the basics of the language in the book. On the contrary, this coverage will put C++ in clearer focus, and Lippman's analysis will probably clarify features of C for them.

I highly recommend *C++ Primer*. It is useful both as a learning tool and as a reference for C++. It is the most cohesive treatment of the language and the most comprehensive treatment of release 2.0 currently on the market. I consider it to be a "must" on your C++ reference shelf.

Reviewed by **Karen Ryer Cunningham**, *a consultant specializing in object-oriented training and software development. She may be reached at arch (503)626-4119*

- Creating a display-only mock-up of reports and screens
- Always rewriting a prototype in a 3GL language for purposes of implementation performance
- Not having advanced system features

In object-oriented systems, more ambitious prototyping efforts are feasible. Besides prototyping screen layout and screen dynamics, it is also possible to develop the underlying software architecture. Individual functionality can be tested in the context of this underlying architecture, so as to study functional interactions, notably the flow of processes and data. Object-oriented prototyping more often involves programming in the equivalent of 3GL code, although, for the most part, this code serves the purpose of combining existing object descriptions from a library to create the new application. Object-oriented technology supports good screen presentation and editing of the presentations, so that tools like 4GL screen layouts can be made available. Currently available interface builders for object-oriented languages, however, fail to include the sophistication required to describe fully the dependency relationships among objects representing information models, and those representing graphical presentations.

What Supports Prototyping?

Prototyping is certainly not a new software development technique. Much has been written about it, and a traditional development model has been devised with prototyping in mind (for example, the Spiral Model [1]). Thus, it is possible that some organizations currently utilize a traditional 4GL form of software prototyping.

These organizations can still expect to benefit from object-oriented prototyping. Why?

The answer to this question lies in the properties, techniques, tools, and libraries of object-oriented technology. The properties of interest are: encapsulation, inheritance, and polymorphism (when implemented in terms of dynamic binding). Utilizing these properties are a variety of design techniques and tools for building systems. Combining these techniques and tools with a well-developed library of reusable components yields a powerful prototyping system.

Encapsulation is the combining of information, and operations on information, into a single entity (the object). Encapsulation is a property that supports the development of modular systems. In such modular systems, changes can rapidly be made to explore new ideas as the prototype evolves.

Object-oriented design utilizes encapsulation by focusing on that which is occurring in the world to be captured, as opposed to being concerned about the details of the implementation. Concentrating more on "what" needs to be done, rather than "how" to do it, means that designers are, in fact, focusing on the specification of the system. Designers are free to be somewhat more cavalier about the implementation algorithms. Moreover, when the object is the unit of encapsulation, the specification is expressed in terms that are closely aligned with the real-world understanding of the problem.

The effort required to create a prototype should be directly proportional to the difference between that which one wants and that which is already available. Inheritance is a technique by which an object is described in terms of another object. Object-oriented design utilizes inheritance in order to identify small differences between kinds of objects. Inheritance is one technique that allows designers to exploit the use of objects which are available from a library, in order to create new functionality, as a refinement of existing functionality. A key to object-oriented rapid prototyping is, therefore, the richness of the available repository of reusable components. As an additional benefit, the resulting prototype is proportionally more robust when it is constructed from software components that are shared and tested by many programmers, than it is when every component is recreated. In our experience, the understanding of which objects should be part of a corporation's library of reusable components often occurs during prototyping.

Polymorphism, when implemented in terms of dynamic binding, also strongly supports prototyping activities. The following example illustrates this.

Suppose you are creating a graphics package made up of various kinds of shapes. Imagine that the prototype now exists, having been constructed from existing objects found in the repository. But the functionality is far from complete—the initial prototype only specified rectangles, triangles, and circles. Because you are a good object-oriented designer, each of the existing shapes was appropriately described in terms of an object that implements common operations, such as draw, erase, or rotate. The object-oriented property of polymorphism means that each kind of shape (rectangle, triangle, circle) can provide appropriate and separate implementations of these operations.

Now you are ready to tackle tetrahedrons and rhombi. How much effort is needed to modify the prototype? In order to add the new shapes, tetrahedron and rhombus objects are described, which have the ability to carry out the standard shape operations of draw, erase, and rotate. In this case, we have modified the system by adding new objects that accommodate a change in specification. It is, however, important to understand that which we did not modify. We added new shape objects to the system, but all of the existing code that utilized shape objects remained the same, and required no recompilation (even though the system now handles tetrahedrons and rhombi). This increased capability, without changing the system users of shapes, was achieved through polymorphism and dynamic binding.

Prototyping is even better supported when done in the context of a fully integrated development environment, so that additions, deletions, and other revisions can be properly coordinated and completed in real time. Such a development environment should provide incremental compilation, symbolic debugging, and various ways to inspect objects, source code, and execution state. Because, by nature, the prototype is going to change very rapidly over a short amount of time, the environment should provide automated versioning and change control as well.

One other feature is useful for prototyping, which is exception handling. Good exception handling is usually thought of as a requirement for production quality systems. Nevertheless, good exception handling is also a requirement of prototyping, in order to support user experimentation with systems that are incomplete, but that are sufficiently robust to give to the potential users. An incomplete system given to a user must degrade gracefully, rather than fall apart when something unexpected happens.

Fitting Prototyping into the Development Process

Over the last decade, our understanding of the software development lifecycle has evolved, beginning with the idea of a directed, top-down process of structured analysis and design, followed by coding, testing, and maintenance, to a mixture of design techniques. We no longer adhere to the strictly linear waterfall approach, in which analysis created a nonchangeable formal requirements specification, and design mapped those requirements to an implementation plan, which was then followed. An initial change in this approach was in making iteration a fundamental part of the prototyping process. This allowed design and implementation efforts to feed back to, and possibly change, the requirements specification. In the mid 1980s, Barry Boehm, one of the most respected advocates of the waterfall model, introduced the spiral model, in which prototyping plays a key role. Simply stated, in the spiral model, prototyping is used to allow for experimentation and testing of aspects of a system in order to allow incremental decision-making.

Prototyping is usually concentrated upon in the early stages of software development. When starting a project, an Analysis Prototype is created, in order to clarify the formal requirements specification. This prototype allows target users to interact with an executable, but partial, mock-up of the expected product, in order to solicit user reactions and ideas. As soon as agreement is reached, a formal requirements specification can be documented, based upon the Analysis Prototype.

At the second stage of development, a Design Prototype is developed in order to create and to verify the system's architecture. This Design Prototype forms the basis for performance and space evaluation, checking for redundancies or inconsistencies in the design, and determining whether functionality can be well integrated. Performance analysis of the Design Prototype identifies, for example, where engineering efforts are needed, before the product version can be completed. The resulting architectural clarity of the Design Prototype provides the basis for coding the actual product software.

Because the Analysis Prototype was executable, it possibly represented a basis upon which to evolve the Design Prototype. More often, however, it is better to discard the Analysis Prototype code, and to start over, with full knowledge of the requirements. Alternatively, the Design Prototype code could, in fact, evolve into the product software, since the design had to account for all aspects of the final product, including integration with standard system software. We have heard it said that good performance and space engineering is best handled by discarding the Design Prototype and beginning again. We have seen both cases. The choice to evolve the prototype into product, however, is primarily dependent on the resulting performance and space analysis of the prototype.

Object-oriented rapid prototyping, as a key part of the software development process, has had mixed reviews for two reasons. First, there is a tendency to be excited when the anticipated application first works, even though the application is, in fact, a first prototype. Hoping to beat competitors to the marketplace, a manager might be tempted to deliver the prototype.

Second, the processes never seem to converge. Rather than one Analysis Prototype, there are two, three, or more; rather than one Design Prototype, there are several. The development team is usually small, so that the product is developed by the same people who did the prototyping. The fact that there is usually a small development team adds to the confusion as to which stage of development the project is in, and whether or not the final product will ever be delivered. In larger companies, a preferred approach would be to make certain that the product coders would not be the same people as the prototypers would be. It would also be preferred that deliverables from the prototyping process would include excellent design documents.

Object-oriented technology provides properties, techniques, and tools targeted for ease in changing a software system. With change so easy, there is unfortunately a tendency to improve constantly the prototype or product. The answer to obtaining convergence at each step of the development process involves an explicit agreement that a prototype is being built. This agreement should include a clear statement of how one intends to evaluate the prototype. Second, a single person should be assigned clear authority to declare the prototype completed, and to carry out the evaluation.

One view of an object-oriented prototype is that it typically consists of an application context, or framework, and a library of objects that will interact in this framework. One technique for controlling the development of a prototype is to make the following individual team assignments:

- Programmer of the prototype framework
- Programmer of objects that are supported by the framework
- Programmer/tester of whether these elements have been designed for broader functionality
- Project leader to read and review on-line documentation and code, and to write the explanatory documentation

The intention here is that a different person will be responsible for each role.

With good on-line documentation, a computer program can be written to generate simply most of the hardcopy reference manual. The project leader is given full authority for assigning daily tasks, and for determining whether the objectives of the prototype have been met.

How is a Prototype Evaluated?

Prototypes produced at differing stages of development are of course, evaluated differently . The purpose of the Analysis Prototype is to obtain the formal requirements specification. Thus, the Analysis Prototype is evaluated upon whether or not it provides a clear basis for requirements validation. In using a 4GL for prototyping, the expectation is that the display screen presentation and interaction will provide the target users with a clear visual explanation of the product idea. In using an object-oriented system for prototyping, however, the expectation is larger. With an Analysis Prototype, a full behavioral specification of the application's information model, as well as the interaction model, can be demonstrated. This behavioral specification is judged upon whether or not it lays out clearly to the user the functionality requirements and any data or state information. One should keep in mind that each function does not have to be actually implemented.

THERE WERE a number of articles referring to the the database area, discussing the pros and cons of OODBMSs.

The "Great Database Debate: Object vs. Relational Model" was described in *Digital Review* (Dec 18, 1989). "'The real world is object-oriented,' said Peter Page, executive vice president of Software AG, a Reston, VA based database management system developer, during a speech at the Database World Conference in Boston, earlier this month. The dominance of object-oriented programming is, in his words, 'inevitable.'" On the other hand "Ignoring the rules for relational database design 'is the most ridiculous thing I ever heard,' said Roberta Rousseau, vice president of educational services at Codd & Date, E.F. Codd's San Jose, California, consulting company. 'That's the cry of the hacker. The information that comes out of these systems is what business decisions are made from, and the database system is responsible for maintaining the purity of that information,' Rousseau said."

❦

DATABASE NEWS (December 1989) had a number of articles exploring OODBMSs. An article by Larry English gives a good overview of the topic, including some definitions and, layered view of levels of "object-orientedness."

The issue of *Database News* also had a long and wide-ranging interview with Tim Andrews of Ontologic, Inc. It covered many of the theoretical, practical, and market related issues of OODBMSs.

Finally, we will quote the editor of the *Database News* issue, who has some reservations about objects: "But commercially-oriented information systems may not be like [objects that have life]. 'Orders,' for example, don't spontaneously change state—instead, they must follow specific rules that strictly govern their update behavior. Nor do customers spontaneously change state—or at least the life options they do have are totally dependent on the structuring rules of the business. They are not independent objects acting on their own (that would be chaos), but rather points of information subject to the external, controlling apparatus of the business." The article continues by questioning if OO techniques are useful in rule-based decision making.

Regularly appearing in HOOT, this section introduces you to companies offering various products and services in the object-oriented field. Every company which sells an object-oriented product or service (old or new) will eventually appear in the Company Showcase.

Included here is a brief description of the company and its product or service, atributes, with name and phone listing for further contact. This section is provided as an information service to readers and no endorsements by HOOT are implied. Summaries are provided by the companies. Companies wishing to be included in the Company Showcase should send appropriate information to Jeff McKenna, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062. Tel. (503) 692-1889

Jade Simulations International Corporation

Jade Simulations International Corporation is the leading developer of simulation software for distributed and parallel computers. *Sim++*, Jade's simulation environment, is specifically designed to support the development of object-oriented, discrete-event simulations. *Sim++* simulations are developed in a sequential environment, and are capable of executing in parallel on a network of workstations or on a multicomputer. Parallel execution reduces simulation time and permits larger simulations to be run.

Sim++ is a C++ simulation library and it includes discrete-event simulation primitives, random number generators, distributed input and output facilities, and a suite of performance analysis tools.

For more information contact:
Lindsay Moir
Jade Simulations
International Corporation
80-1833 Crowchild Trail N.W.
Calgary, Alberta, Canada
T2M 4S7
voice: (403) 282-5711
fax: (403) 289-6876

Peritus International

Peritus International, founded in 1983, is a systems software company specializing in the provision of high performance C/C++ and FORTRAN compiler technology for OEM manufacturer systems.

Since its inception, Peritus's goal has been to develop highly optimized, portable-retargetable compilers and related development tools. Peritus has successfully met this objective with the development and subsequent release of state-or-the-art compiler technology. This technology is highly portable and it encompasses a wide range of optimizations, some of which are not available in any other present-day, production-quality compilers. Peritus continues to expand its hardware/operating system support, and to improve its compiler technology, through extensive ongoing research and development efforts.

The Silicon Valley-based firm, headquartered in Cupertino, California, is headed by Ronald Price, president. Mr. Price brings experience with both corporate and engineering management from a number of software product companies, including Microtec Research, OMEX, and Spectra Medical Systems.

The Peritus team also includes two founders of the company, who bring significant technical expertise and experience. Reza Yazdani was previously a senior member of the technical team at CDC, developing the FORTRAN compiler for the CDC 205 supercomputer. Sassan Hazeghi authored the Stanford Pascal370 compiler, and previously managed compiler development at Microtec Research. Both founders have been key technical contributors to compilers delivered to CDC/ETA Systems, Apple, Concurrent, and Amdahl. Yazdani and Hazeghi lead technical operations as vice president of engineering and vice president of research and development, respectively.

The technical team at Peritus has achieved some significant accomplishments in the relatively short time since its inception. To date, Peritus has implemented a production-quality superset FORTRAN compiler, a state-of-the-art C/C++ compiler, and a sophisticated source-level debugger. These products have been customized for a variety of proprietary hardware architectures and operating systems under major OEM agreements. The Peritus team includes skilled software engineers with production experience in the design and implementation of large system software.

For more information contact:
Ronald Price
Peritus International, Inc.
10201 Torre Ave., Suite 295
Cupertino, CA 95014
voice: (408) 725-0882

The monthly influx of articles appearing in the trade press shifted its distribution slightly last month. Since it was the end of a decade there were many commentary articles that discussed the emergence of OOT in the 1980s, and the prediction that OOT would become even more important in the 1990s. Approximately 15% of the articles that appeared were commentaries; some were of this retrospective nature and many other commented on OOT as being "good" and "desirable," and also a few articles that discounted OOT as the latest hype. The category of articles that were of an informational nature, introducing readers to OO concepts grew over the past months. Last month some 30% of the articles were in this category. We also saw product announcements drop to 18% of the articles. The "hype" articles, which used the term "OOT" for marketing impact, remained at about 10%.

ON THE PREDICTION front, even dailies are getting into the act. The *San Francisco Chronicle* (Nov 27, 1989) predicted that OOT would be a major factor in software development in the 1990s.

Infoworld (Jan 1, 1990) had some discussion, which follows, regarding the future of OOT. "'The 90s are going to see an accelerating use of OOP,' Intersimone [David Intersimone of Borland] predicted. He noted that mainstream OOP tools have begun to emerge, and that an ANSI committee to standardize C++ held its first meeting in December. . . . 'The coming changes will not occur overnight, however. The notational changes of the last decade have made a big difference in saving programmers' time,' Orr [Mike Orr, previously of Microsoft] said. '[But] I still program in terms of integers and floating-point numbers and, as long as that's true, it's still going to be nine days of mechanical grinding for one day of creative thinking.' Orr further said, 'Any such improvement takes a long time to absorb and sets you back at the beginning, [but] OOP does have the potential to be really revolutionary.'"

And last, *Communications Week* (Dec 25, 1989) had a short wish list for OOT: "Finally. For vendors of non-object-oriented products; an excuse to call their products object-oriented. For vendors of object-oriented products; someone who can explain object-orientation. And to all, a good '90."

❦

WE FOUND a few articles that described some ongoing projects using OOT, and described some of the observed benefits.

In the *Financial Times Daily* (Nov 24, 1989) we found a good overview article that described some developers and users of OOT: "One company, Cadre Technologies, a computer aided software engineering enterprise based in Providence, Rhode Island, reckons to have reduced the amount of code needed for a new application by a factor of 5:1, through the reuse of existing OOPS modules."

In *Infoworld* (January, 15, 1990) there was a brief article on the use of OOT by Aldus Corporation and Eikon Systems. The article also says: " Other companies have not been so bold, however. For example, Redmond, Washington-based Microsoft Corporation—which last February announced a comprehensive strategy to implement OOP across its systems, languages, and applications—is still using OOP for "less than 10 percent" of its work, according to Microsoft CEO Bill Gates.

"We're using OOP internally, but to a limited degree," Gates said, adding that most of that work is being done using C++ 2.0. Microsoft and Borland International both rolled out object-oriented Pascals last spring and have promised to release versions of C++.

"Borland has opted to use object-oriented techniques in developing its applications but has not yet begun using OOP languages in release versions," said Brad Silverberg, vice president of research and development at the Scotts Valley, California, company. The company used OOP techniques to develop its Virtual Real-Time Object-Oriented Memory Manager, which is used in current versions of Reflex and Quattro Pro..."We found [using OOP techniques] has tremendously improved our productivity," Silverberg said. "It reflects the way people really work—they start with something that works and make incremental changes to it."

In *UNIX Today* (Dec 11, 1989) the use of OOT in the development of the FAA's Automatic Terminal Information Service was described. The combination voice and data system is explained, and the developers cite the use of OOT in rapidly developing fully functional prototypes in 18 months as being a major win.

Network World (Dec. 18, 1989) described the use of an object-oriented operating system (Alpha) for use in SDI by the Air Force. "...the Air Force is interested in Alpha mainly because it will lower the cost of developing distributed applications..." The developer is Concurrent Computer Corporation.

The January issue of *System Builder* discussed reusable software in its productivity section. The article discusses the differences in reuse when using traditional techniques (libraries) and when using objects. It also points strongly toward the Japanese. "Japanese companies reward software reuse and often set targets for new code development. For example, new systems must create a certain percentage of new reusable modules for the library. Project target dates cannot be met if all code is developed from scratch—they force the reuse of existing code."

❦

THERE WERE further comments on the emerging debate on standardization. In *Computer Design* (December 11, 1989) Tom Williams says: "So C and Pascal are likely to steal the object-oriented show away from the languages that first made object-oriented programming popular. It hardly matters that Smalltalk aficionados can prove that their language is "better." Standards will always carry the day.

"Take C as an example. It has surpassed Pascal in popularity. Part of its success comes from its versatility, but C can credit much of its popularity to the incompatible dialects that plagued Pascal in its early days. Though that's now changed, the early acceptance of Pascal was hampered by the compatibility issue. C, on the other hand, has consistently offered programmers and compiler designers a significant advantage—a clear and distinct ANSI standard."

❦

A FUN ARTICLE by Jeff Duntemann in *Dr. Dobbs Journal* (January, 1990) described some "Rubber Truths" about OOT. His view of the role of hierarchy is overstated, but it does point out areas of concern. He comments that bugs, as well as features, are inherited, and that the quality of the reusable parts is very important.

The Design Prototype demonstrates the feasibility of a particular implementation architecture. Both kinds of prototypes (Analysis and Design) should also be examined in order to identify the potential for creating reusable components to be added to the corporation's library. Moreover, the corporation's library itself can be evaluated for how well it supported the prototyping process.

In summary, management should expect the following from the careful inclusion of both Analysis and Design Prototypes in the software development process:

- Decision-making support,that is, whether or not to proceed with a product project
- Improved design understanding and documentation
- Improved results from the end user perspective
- Probably no change in the time for completing the product
- Improved repository of reusable software components

The addition of rapid prototyping as part of the development process changes three things:

- Code is often developed that has to be discarded
- Programmers are assigned as "professional prototypers"
- Development of reusable components is improved, because purposeful attention is paid to the development, testing, and evaluation of such components

Myths of Old

Object-oriented technology breaks many of the myths about prototyping that came, of necessity, with the more traditional software development tools. From our experience, it is not the case that:

- Throwing away code is bad (it was only bad to discard code that used too much time and energy to create in the first place)
- Only display screens are easy to prototype (use of object-oriented techniques for composition and refinement make it possible to prototype rapidly at all levels of a system).
- The language for prototyping is necessarily different from the production programming language (that is, an object-oriented 3GL language can serve as a prototyping language as well as the primarily screen-painting 4GL languages).
- Performance of the prototype does not matter (it certainly does, if the prototype is to be used in target-user studies).

In the next article, we will describe some case studies in prototyping, only one of which actually succeeded, in order to begin to examine workable ways in which to structure the team and team activities.

Bibliography

[1] Boehm, B. W., "A Spiral Model of Development and Enhancement," Software Engineering Notes, Vol. 11, No. 4, August, 1986.

[2] Connell and Shafer, *Structured Rapid Prototyping*, Yourdon Press, Prentice Hall, 1989.

SCOOP-West and the Software Industrial Revolution

by Jeff McKenna

The latest incarnation of SCOOP, SCOOP-West, was held in Santa Clara, California, at Techmart, January 22-24, 1990. Following the successful launch of SCOOP last year in Boston, approximately 400 programmers and managers from Silicon Valley attended SCOOP-West this year. The conference consisted of 15 tutorials, four panels, an invited lecture, an exhibit area with 33 vendors, and product presentations by the vendors.

The keynote speaker was by Philippe Kahn, chairman, president, and CEO of Borland Corporation, the producers of Turbo Pascal. Philippe first explored the possibility that OOP may be just a buzzword. He used the history of Artificial Intelligence over the last decade as a model of one possible path for the future of OOP. While noting that OOP really works, he emphasized that the perception of OOP as a niche needs to be refuted. His main point was that the introduction of widely accepted "main-stream" programming languages that support OOP, including Turbo Pascal (surprise, surprise!), is the key to the widespread acceptance of OOP. In responding to questions from the audience, he emphasized that OOP is immature, and many questions remain as to the details. This is especially true in the front end of software development. The most interesting question asked was whether Borland used OOP internally in its own development. Philippe responded with an emphatic YES!

Borland was noticeably present at the Reception as well, with the Borland Turbo Jazz Band, including Philippe, playing some loud music and giving away some colorful T-shirts!

The tutorial program was the cornerstone of SCOOP and most sessions were very well attended. There were sessions on management issues, OO design, OO databases, and applying OOP to graphical applications. There were also tutorials on specific languages such as Smalltalk, Eiffel, C++, Objective-C, and CLOS. The attendees I talked to felt that they were getting good exposure to the ideas and practicalities of using OOP.

The low-key exhibit area gave attendees the opportunity to see the latest in product and service offerings. There were a few new companies, but most of the vendors were familiar to the readers of *HOOT*. The people involved with the object-oriented database appeared to be very busy, since Object-Sciences is becoming more public and they have a working product.

ExperTelligence announced their "Action!—Oriented Programming™" for the Macintosh. This product was very impressive, giving the user a powerful tool for application development. Peritus introduced its new C/C++ compiler. (See the product introductions in the Product News Section). There were also two new Smalltalk development packages, announced by Zuniq, Inc. and Object Technology, Inc.

SCOOP-West included panels on OO databases, the packaging of OOP products, and OOP and visual programming. These panels provided various points of view to the attendees. SCOOP also included the mandatory panel on Language Wars. The conclusion: use the most appropriate language!

The most interesting session for me was the invited lecture by Brad Cox of The Stepstone Corporation. This lecture was entitled "Planning the Software Industrial Revolution—The Impact of Object-Oriented Technology." A summary of this talk follows, augmented by other notes that Brad passed to me.

Brad has been exploring the explosive development in productivity that accompanied that which we commonly refer to as the industrial revolution. He speculates that the software industry is poised on the edge of a similar breakthrough. By studying the case histories of the industrial revolution, Brad has identified various conditions that stimulated this historical breakthrough.

The most interesting condition is that the customers, not the producers, forced the development. The requirements of the customers demanded that the producers improve their processes in order to deliver products in sufficient quantity and quality. We see parallels in today's software world. The story of Ada is an example of a situation in which customers demanded better products from the producers. The current "demand" for standards is a similar situation.

The next condition stimulating the breakthrough of the industrial revolution was the development of standard, interchangeable parts. As Brad noted, this took quite some time, and the effort and was resisted by the producers of the day! Stepstone's concept of the software-IC, developed by Brad, is an example of how the software industry is attempting to define these standard, interchangeable parts. We see other activity—windowing systems or SQL, for example, defining an interchangeable software "part," as well.

The last condition of major importance in the development of the industrial revolution was the emergence of specification and testing tools (blueprints, gauges, micrometers, etc.). These tools were used to test for acceptability of the standard, interchangeable parts. They were also used to determine conformity with the specifications. In today's software world, this condition is not well supported. In his papers, Brad challenges the software industry to address and solve this pressing problem.

Brad points out that the major changes that enabled the Industrial Revolution to come about were cultural, not technical in nature. The majority of the producers prior to the revolution did not participate in it. He cautions that a very similar event could happen in the software industry. We could drift away, debating the merits of this or that language, or this or that operating system, while the customer solves his problems without us.

All in all, this was a very interesting lecture. I also want to thank Brad for a good deal of discussion about this topic at the coffee table.

That is about it for SCOOP-West. It was good conference with plenty of enthusiastic attendees, and a rich program focused on the practical applications of OOP.

Australian Object-Oriented Scene

by Dr. B. Henderson-Sellers

Australia is a large country. It is as far from Perth to Sydney as it is from San Francisco to New York. Consequently, it is virtually impossible to avoid parochial biases, and to provide an overall synopsis of the object-oriented scene across the whole country of Australia. Nevertheless, it is my intention in future articles to reflect object-oriented usage and developments in other centers in Australia, although this first report shows a Sydney bias.

There are a small number of industries that have been developing object-oriented systems for a few years (mostly in C++ and EIFFEL—in addition to those using Ada). The vast majority of commercial and academic organizations associated with object-oriented developments, however, appear to be relative newcomers. Industrial IS departments are keen to acquire knowledge about the OO paradigm, especially in respect to OO analysis and design methodologies, and their implications for their own departments. The departments are particularly interested in the costs/benefits in adopting OO technologies. Yet they are also concerned with their current (often very recent!) investment in functionally-based software tools, such as front-end and back-end CASE tools. Consequently the dilemma in embracing object-orientation is whether or not these existing tools can be adapted in some way. Those industries who attempt to adapt these tools often try to "mix-and-match" object-oriented ideas with existing functional decomposition techniques, which produces a hybrid life cycle development methodology.

Research is certainly being undertaken, both in academia and in local industries in Australia, to consolidate methodologies for object-oriented systems development, in terms of both hybrid methodologies (such as the Functional analysis, Object-oriented design, and Object-oriented implementation (or FOO)) as well as the pure OOO (Object-oriented analysis, Object-oriented design, Object-oriented implementation) approach. Other variants such as OOF are also being researched.

In teaching, there are two very active and rapidly growing areas: (i) university courses, and (ii) short courses for professional in-service training. Despite the often lengthy lead times in initiating University courses, several Universities in Sydney and Melbourne have already begun teaching object-oriented programming, either at the Master's level or as a basic undergraduate language. In several cases EIFFEL can be taught as a first language whereas variants on C, such C++ or Objective C, first require a grounding in C.

Several companies have been started in the last two years by people committed to the object-oriented approach. These companies are running short (1-2 day) courses for specific industrial groups. The courses are usually comprised of a one day introduction to the ideas, and a one day description the elements of a particular language (e.g., Smalltalk, C++). Other companies and universities are currently advertising longer, language-specific, hands-on courses. The concepts of object-oriented analysis and design, still evolving, are also beginning to be taught at these levels, and these concepts are elements of the two day courses. The Australian Computer Society is also likely to begin to run object-oriented seminars in mid–1990.

One of the highlights of the past six months for many object-oriented practitioners, and would-be practitioners, was the visit of Bertrand Meyer, who presented his two day seminar in Sydney. About 200 information systems and computer science professionals attended the seminar.

Overall, the commercial world is rapidly "waking up" to object-oriented programming and system development methodologies. This world is beginning to demand further information. It is currently a very exciting time in Australia to be involved in object-oriented systems.

Dr. B. Henderson-Sellers *is a Senior Lecturer at the University of New South Wales in Australia. He may be reached at the University of New South Wales, P.O. Box 1, Kensington, New South Wales, Australia 2033. Telephone: (02) 697 4413.*

Calendar

The HOOT calendar presents conferences and meetings that are focused exclusively on object-oriented technology. It should be noted that many other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to the Editor, HOOT, 8825 SW. Umatilla St., Tualatin, OR 97062

MARCH 13-15, 1990

The Object-Oriented Symposium—Presented by Digital Consulting, Inc. and chaired by Larry L. Constantine in Washington, D.C. Tutorials and exhibits. Contact: DCI, 6 Windsor St., Andover, MA 01810, (508) 470-3880.

APRIL 9-11, 1990

2nd USENIX Conference on C++—San Francisco, CA. Tutorials, technical sessions. Conference chair: Jim Waldo. Keynote: Adele Goldberg. Contact: USENIX Conference Office, 22672 Lambert St., Suite 613, El Toro, CA 92630, (714) 588-8649.

MAY 21-24, 1990

SCOOP-East—Wang Institute of Boston University. Co-sponsored by JOOP and BUEC. Tutorials, Panels, Product Presentations and Exhibits. Keynote: Niels Jensen, JPI. Contact: Bob Daniels, BUEC, 72 Tyng Rd., Tyngsboro, MA 01879, (508) 649-9731.

JUNE 6-8, 1990

Eurographics Workshop on Object-Oriented Graphics—Konigswinter, Federal Republic of Germany. Workshops cover standardization and OO languages for graphics. Contact: Marja Hegt, O-O Graphics Workshop, CWI, Kruislaan 413, 1098 SJ. Amsterdam, The Netherlands. Tel. 31-20-592-4058.

JUNE 26-29, 1990

TOOLS '90— Second International Conference and Exhibition. CNIT-Paris, France. Program chair: Jean Bezium. Conference Chair: Bertrand Meyer. Tutorials, workshops, technical sessions. Contact: TOOLS '90, SOL, 14, rue Jenn Rey 75015 Paris, France. Tel. 33-1-40-56 0358.

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes,*** I want to be plugged into the latest thinking and developments in the object-oriented technology. Immediately enter me as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	**2 years (24 issues)**
☐ ~~299~~ $249	☐ ~~$598~~ $498

outside the U.S.—add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT—U.S.currency drawn on U.S. bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card #_____ Exp. date_____
Signature _____

Name _____
Company _____
Address _____
City _____ State_____ Zip_____
Phone_____

For immediate service—call the Hotline 212-274-0640

HOTLINE on
Object-Oriented Technology
The International Newsletter

588 Broadway
Suite 604
New York, NY 10012

HOTLINE on
Object-Oriented Technology
The International Newsletter

April, 1990 **Volume 1, Number 6**

Life-Cycle Approaches

by Edward V. Berard

Sometimes, to an outside observer, it appears that software is developed in a process that resembles Brownian motion. Unfortunately, it is true that many software "engineering" efforts are little more than organized chaos. This may be due to ignorance, poor management, lack of training, or any one of a number of other reasons.

A professional is someone who makes his or her task look easy. From a virtual infinity of choices, a professional seems to always choose the most appropriate next step in the process at hand. There is very little wasted motion. Even "trial and error" efforts appear to progress deliberately towards an effective resolution.

Unlike the "new thinkers" in George Orwell's 1984, professionals do not believe that "ignorance is strength." A professional knows that knowledge, methods, and discipline are not a "strait jacket," but instead amplify his or her creative capabilities.

Before we take a detailed look at object-oriented design or object-oriented requirements analysis, it might be a good idea to look at life-cycle approaches in general. If we understand how people have tackled software engineering projects in the past, we will be better able to discuss and understand the object-oriented life-cycle.

Methodologies And Life-Cycle Approaches Are Not Always Necessary

There are times when an ad hoc software creation/development process is justifiable. In fact, in some circumstances a (semi-)rigorous methodology may be worse than no methodology at all. The trick is to know when and where to consider using a methodology.

If your project is small, or non-critical, or both, a methodology may not be important. Let's look at "small" first.

A number of factors can effect the "size" of an effort. If we are only talking about less than 2,000 lines of code in just about any programming language, that may be considered small. However, if we get much above this limit, things change. In 1976, F. DeRemer and H. H. Kron published an article (F. DeRemer and H. H. Kron, "Programming-in-the-Large versus Programming-in-the-Small," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 2, June 1976) which gave a name to an important concept: "programming in the large."

People had long suspected that a 100,000 lines of code project was not simply 10 times the effort of a 10,000 lines of code project. As it turns out the effort required to produce the larger project may be 50-75 times the effort required to produce the smaller project, not merely 10 times the effort. The factors which contribute to this massive, and "unexpected" increase in effort include:

- error density increases with overall size, i.e., the rate of errors per unit of code increases with the total amount of code written. As it turns out, this problem can be mitigated by reusing software, and, thus, reducing the total amount of new code that has to be written. [Yes, software reusability has a positive impact on reliability.]

continued on page 3

Contents

Feature:
Life Cycle Approaches
1

Thoughts from the Editor
2

Databases
A Data Manipulation Language for C++ Developers—A Proposal
5

Project Management
Case Studies in Prototyping (part 1)
6

Analysis & Design
I Already Learned in Kindergarten All I Need to Know About Object-Oriented Design
9

Calendar
11

Publicity Corner
11

Book Review
Readings in Object-Oriented Database Systems
14

Product News
15

In this month's issue of the Hotline, the feature article by Ed Berard is on software life-cycles. In the article, Ed gives a good overview of many of the different life cycle models from the point of view of objects. This is an dynamic area with a great deal of activity on both the theoretical and practical fronts. The whole topic currently poses more questions than it answers.

Adele Goldberg and Kenny Rubin continue their series with an exploration of software prototyping (mentioned in Ed's article as well!) practice. They explore some successes and failures in using prototyping.

Tom Atwood continues his database series by proposing a database manipulation language for OODBMs. He challenges the OODB community to design/implement a standard interface language, similar in purpose to the SQL language of relational databases.

Alan Purdy discusses some of the same issues is in a review of a recent book on OODBs.

The Publicity Corner has expanded this month. Molly Faherty of Regis McKenna, Inc., (no, we are not related!) is providing the Hotline with a more comprehensive review of each month's clippings.

Ed Berard's feature article got me to thinking about the state of OO Analysis and Design. Customers have been demanding a methodology for doing analysis and design. I don't think this is going to happen for quite a while. We are now seeing various proposals (Booch, Shlaer and Mellor, Coad, Page-Jones and Weiss, and others) that are all being used in practice. I don't see (or want to see) at this stage a single methodology suddenly rising up as the "best". The industry needs time to try out these techniques, distill the essence of what OO Analysis and Design really are, and formulate methodologies to support this essence.

A proliferation of "circles and arcs" based products with no discernible internal content is not what the industry needs. I see this as the primary weakness of many (not all!) of the current CASE offerings. I hope that OO developers are not caught in the same trap.

With the use of objects, one can more clearly see what are the real issues of software development since the effort of coding is considerably reduced. I hope to see, and am seeing, widely differing ideas and products that will help us discover those issues and point us toward effective solutions.

Jeff McKenna
Editor

HOTLINE™
Object-Oriented Technology
The International Newsletter

Jeff McKenna
EDITOR

ADVISORY BOARD
Jim Anderson
Digitalk, Inc.
Tom Atwood
Object Design, Inc.
Grady Booch
Rational
Larry Constantine
Consultant
Cort DeVoe
IBM
Adele Goldberg
ParcPlace Systems
Bertrand Meyer
Interactive Software Engineering
Reed Phillips
Knowledge Systems Corp.
Trygve Reenskaug
Taskon A/S
Bjarne Stroustrup
AT&T Labs
Dave Thomas
Object Technology International
Zack Urlocker
The Whitewater Group
Eugene Wang
Borland International

Richard P. Friedman
Group Publisher
Margherita Monck
General Manager
Xavier Yaffar
Technical Advisor
Melanie Younossi
Marketing Manager

The Hotline on Object-Oriented Technology (ISSN#1044-4319) is published monthly by SIGS Publications Inc., 588 Broadway, Suite 604, New York, NY 10012. (212) 274-0640. ©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express persmission from the publisher.

Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30. Send address changes and subscription orders to HOOT, 310 Madison Avenue, Suite 503, NY, NY 10017
Submit editorial correspondance to Editor, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062, Tel. (503) 692-1889.

mative that you feel satisfied without actually reading the papers in-between. Oh, I know, some of you purists insist on interpreting history directly from the primary sources. Well, this compendium should please you too: the relevant historical record is all there. For us others, we can rest assured: Stan and Dave are primary sources - they have each blazed trails at the OODB frontier. More importantly, like the troubadours of old, they interpret and distill the stories of the object-oriented revolution, and clearly sing fascinating songs for us all.

Readings in Object-Oriented Database Systems begins with a finely crafted introduction to object-oriented database systems. This introduction describes the forces motivating OODBMS development, defines the terms, distinguishes consensus from controversy, links the movement to the record-oriented world it promises to encapsulate, proposes both a threshold and a reference model upon which to evaluate various research vehicles and products, and finally, predicts where the OODB field is headed. This introductory chapter alone is worth the price of the book.

In the introduction, Zdonik and Maier's threshold model defines "the minimal requirements for a system to be called an object-oriented DBMS." Their reference model defines the "... capabilities that a competitive commercial product might support." By defining these two models, the editors distinguish areas of agreement from areas under active debate in the OODB community. While this segmentation provides interesting insight to the OODB researchers, their explicit avoidance of defining a standard for evaluating OODB products can only prolong the time needed for these products to mature.

The first chapter on object-oriented fundamentals refines further the concepts of the object-oriented model as seen from the frontier: object identity, user-defined types, encapsulation, and inheritance. The remaining chapters present and clarify the major issues all OODB systems developers must address: semantic data models, persistence, implementation alternatives, transactions, change management, relational extensions, interfaces, and applications.

Object-oriented experts may quibble with the editors' choices for the papers selected for inclusion. For example, why include a CLU paper instead of one on Simula-67? Also, the editors predict in the interface chapter that "...the object-oriented database movement will benefit from hypermedia systems." Yet, they don't include any papers that demonstrate how to push the outer envelope on direct manipulation of graphical interfaces such as Ingalls' Fabrik, or even HyperCard and its clones. But these are only quibbles, and experts do enjoy a good quibble.

The OODB marketplace would be better served if an individual with a Codd-like stature guided our field with as steady a hand as Dr. Codd has provided to the relational database movement during the last 20 years. Codd's twelve requirements for relational systems have been the driving force behind both the technology and the products. While several Object Management Group (OMG) task forces are defining standards for OODB systems, industrial committees rarely demonstrate the inspired leadership of an individual. Perhaps we can convince Zdonik and Maier to share a Codd-like role in the shaping of the OODB industry. Their technical mastery of the subject is beyond reproach, and they are not perceived as tied to any particular product.

The beginning of this review is not completely in jest. The book's preface lists various locations where the editors worked while selecting the relevant articles and composing the book's chapter introductions. Surely, these two personify the post-modern jet-set troubadour. They take us from Harvard Square, to a German train, to a hot tub on the Oregon coast. If Stan and Dave do this for a living, this reviewer wants to be invited to their next party.

Reviewed by Alan Purdy, *president of Instantiations, Inc. Alan was one of the early members of Servio-Logic and worked at Xerox PARC Northwest. He was co-chair of the first ACM OOPSLA conference. He may be reached at (503) 242-0725 or 921 SW Washington, #312; Portland, OR 97205.*

Interactive and Data General Announce Eiffel for the 88000 RISC-Based AViiON

Interactive Software Engineering announced that it has signed an Independent Software Vendor (ISV) agreement with Data General Corporation. Under the agreement, Interactive will supply its Eiffel software on Data General's new 88000 RISC-based AViiON co-system under UNIX. Eiffel is an advanced object-oriented language, method and environment designed for the production of high-quality software. Eiffel emphasizes reusability, extendibility, correctness, and robustness.

"Data General AViiON systems running Interactive Systems' object-oriented programming language provides a state-of-the-art platform for the development of engineering quality software for production applications," says Joel Schwartz, vice president of Data General's Software Business Unit.

According to Tom McCarthy, Interactive's vice president of Product Marketing, "Interactive appreciates Data General's firm conviction to offer a broad variety of software applications on a non-proprietary system such as the 88000. We have responded to the same trend by allowing Eiffel to generate stand-alone C packages that can be developed on the AViiON and easily ported to other UNIX platforms."

Data General Corporation is a leading manufacturer of computer systems for business, industry, education, and government. The company reported revenues for the year ending September 30, 1989 of $1.31 billion.

For more information, contact:
Interactive Software Engineering
Thomas P. McCarthy
270 Storke Road, Suite 7
Santa Barbara, California, 93117
(805) 685-1006

..."There aren't many commercially available industrial strength products, "[Steve Haggerty, vice president for systems technology at Citicorp] said. Haggerty thinks OOP's best chance for success might be with companies such as Ingres, Information Builders and Oracle Corp., which develop tools on a variety of platforms and are fairly well-known in the corporate world already.

Source: "Stuck on the Runway," *Computer Reseller News,* Philip Rose, January 15, 1990, pp. 2, 146.

The 1990s will see the rise of design tools that are collected into open environments. More that just graphic interface environments, those sets will include true design tools—development tools that aid designers in describing and monitoring their hardware or software creations. Such tool collections will feature object-oriented software design, with C++ as the first standard OOPS.

Source: "Source Code: Doing it Easier," *Electronic Engineering Times,* Ray Weiss, January 15, 1990.

Object-oriented programming has been celebrated for forcing programmers to write well-structured code, but its greatest potential is in its breaking down of the "superprogram" approach to advanced application programming. This shift away from do-everything applications toward discrete components that may act independently or in concert offers tremendous potential to smaller companies with good ideas.

Source: "More Complex Times Lie Ahead," *Computer Reseller News,* Philip Rose, January 22, 1990, pp. 2, 161.

Industrial-strength products that can satisfy end user desires [such as constructing medium- to large-scale knowledge bases that are maintainable, easily extensible and that reduce the costs associated with software engineering] have long been unavailable. However, in the late 1980s, significant strides have been made in object-oriented programming and representation and in object-oriented database technology to address these industrial desires. Object-oriented programming and representation have surfaced to become far more dominant factors in building knowledge bases than rules-based programming and representation. .It is important to understand that object-oriented programming and representation, in addition to object-oriented database technology, provides the ideal platform to develop intelligent CASE (computer-aided software engineering) tools and facilities.

Source: "1980s Bring Knowledge to the 1990s," *Digital News,* Michael Stock, President, Artificial Intelligence Technologies, January 22, 1990, pp. 23-24,62.

At present, few truly object-oriented CASE products exist. If the definition of CASE products is stretched to include object-oriented programming languages and object-oriented database management systems, the situation improves slightly. Some vendors of more traditional CASE technology have altered their marketing literature—but not their products—to claim that their products support the object-oriented paradigm.

Source: "Understanding Object-Oriented Approaches," *Data Base Newsletter,* Edward V. Berard, February 1990, pp. 1, 77-10.

Standards

OMG (Object Management Group) is defining object-management terms, establishing standards and giving its stamp of approval to standards-compliant products ...during the next 18 months.

Source: "OMG to Wade Through 'Object' Morass," *Systems Integration,* Mary Jo Foley, January 1990.

In 1989, a standard emerged for object-oriented programming and representation that is known as CLOS (Common Lisp Object System). CLOS represents a superset of the functionality of previous object systems based on industrial strength methodologies packaging and programming constructs.

Source: "1980s Bring Knowledge to the 1990s," *Digital News,* Michael Stock, President, Artificial Intelligence Technologies, January 22, 1990, pp. 23-24,62.

Now the Mac world has an object-or ented C from a major software house: Symantec's Think C. (Think C is a subset of C++.) ...Symantec has done such a good job of implementing this language that Think C is likely to become a standard.

Source: "Think C 4.0," *Mac World,* Charles Seiter, February 1990, p. 182.

Book Review

Readings in Object-Oriented Database Systems

By

Stanley Zdonik and David Maier

What do you get when you cross a globe with two chatty database guys? Answer: a grand tourist guide to the entire object-oriented database field. Let's face facts: the refereed scientific literature is boring, boring, boring. How then, can a collection of seminal papers from the OODB literature be anything less than one big yawn? Well, Doctors Zdonik and Maier organized their selected papers into nine chapters, each with an overview so infor-

continued from page 1

- as the number of people assigned to a project grows, so do the number of communication paths. (The total number of communication paths is n*(n-1)/2, where n is the number of people involved in a project.) While communication is necessary, the more communication paths you have:
  - typically the longer it takes to get anything done, and
  - the greater is the chance that something will become corrupted in the communication process.

- the probability and frequency of the "ripple effect" increases with overall systems size. (The "ripple effect" is the introduction of new errors as a result of "fixing" old errors, or "enhancing" the product.)

Size can also be impacted by the choice of programming language. What may have been a good sized project in assembly language, may be a small to medium project in Smalltalk.

Be forewarned, however, even "small" software can be critical software. Gerald Weinberg (G. M. Weinberg, "Kill That Code," *Infosystems*, August 1983) documents an error that cost a company US $1,600,000,000, and was the result of changing a single character in a line of code. Remember "small" is not the same thing as "non-critical."

Software is being used increasingly in applications where reliability and safety are essential. Software is important, and software errors can have serious results, other factors can make the use of a methodology ill-advised, including:

- poor, or non-existent training in the methodology
- the wrong methodology for the type of project, e.g., using classic structured technology on an object-oriented project
- lack of professional guidance
- a poorly-defined methodology
- lack of management commitment to the use of the methodology

We can conclude with these observations:

1. The need for a methodology increases with both the size and critical nature of the software.
2. A given methodology must *decrease* the overall complexity of the software engineering effort — thus increasing the chances of a good quality product. Improper introduction of a methodology, or the use of an inappropriate methodology, can have a negative impact on a project.

General Life-Cycle Approaches

There are three general life-cycle approaches:

- Sequential
- Iterative
- Recursive

In a sequential approach, once one has completed a step, one never returns to that step, or to any step previous to that step. This is only practical with powerful tools (e.g., fourth generation languages) or on very small, non-critical projects.

In an iterative approach, if there is sufficient reason to do so, one may return to a previously completed step, introduce a change, and then propagate the effects of that change forward in the life-cycle. Most of the life-cycle approaches you are probably familiar with are iterative.

A recursive approach is one where the entire approach may be re-applied to the end products of the approach. All recursive life-cycle approaches are iterative, but not all iterative approaches are recursive.

Some Examples of Life-Cycle Approaches

H.A. Kinslow in [P. Naur and B. Randell, Editors, *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*, Garmisch, Germany, October 7-11, 1968], describes the "Flowchart Model":

1. Flowchart until you think you understand the problem ...
2. Write code until you realize that you don't ...
3. Go back and re-do the flowchart ...
4. Write some more code and iterate to what you *feel* is the correct solution.

This approach is still in use today, except that people seldom draw flowcharts.

The "Sequential Waterfall" or "Stagewise Model" assumes that "everything is right the first time." Each step must be "signed off" and it is against the rules to go back or jump ahead.

The most common life-cycle approach today. at least in places where they attempt "formal" life-cycles, is the "Iterative Waterfall" or "Cascade" life-cycle. This approach requires that you complete and entire step, verify the results of the step, and then continue on to the next step. You may, however, at any time, return to some previously completed step, introduce a change, and then propagate the effects of that change.

The Waterfall life-cycle also usually implies that *all* requirements analysis is completed before going on to design, and that *all* design is completed before coding starts. In effect, once a phase is completed, the only time you return to it is *if* you wish to introduce a change.

[The waterfall life-cycle came into existence because of the grossly disorganized attempts at software development that seemed all too common. You know the story: software is late, buggy, much too expensive, and ill-suited to the user's needs. Programmers would "get and idea of what the client wanted," and then proceed to write a bunch of code. Once the coding was done, the paperwork (e.g., design) was attempted — but only until the next project started.]

Birrell and Ould (N.D. Birrell and M.A. Ould, *A Practical Handbook for Software Development*, Cambridge University Press, Cambridge, United Kingdom, 1985) describe the "b-Model." The b-Model recognizes that the maintenance phase of the software life-cycle is seldom just fixing errors. If a software product exists for any period of time, "maintenance" almost always involves enhancements. After a while, the "maintenance phase" of a software product looks very much like the "development phase."

Barry Boehm has popularized the "Spiral" life-cycle (B. W. Boehm, "A Spiral Model of Development and Enhancement," *Software Engineering Notes*, Vol. 11, No. 4, August, 1986). According to Boehm, a "typical cycle of the spiral":

1. Begins with the identification of:
   a. the objectives of the product
   b. the alternative means of implementing this portion of the product
   c. the constraints imposed on the application of the alternatives
2. The alternatives are evaluated. The evaluation may involve "prototyping, simulation, administering user questionaires, analytic modeling, or combination of these and other risk-resolution techniques."
3. The next step may involve "a minimal effort to specify the overall nature of the product, a plan for the next level of prototyping , and a development of a much more detailed prototype ..." The next step may, however, "follow the basic waterfall approach, modified as appropriate to incorporate incremental development."
4. Each cycle is completed by a review, with go/no-go plans made for the next cycle,

As the Spiral life-cycle progresses, the overall cost of the product is determined by the radius of the spiral, and the progress is determined by "angular displacement." The "spiral" moves outward from the center.

We can make two additional observations about the spiral life-cycle:

- The Spiral life-cycle presents an organized approach to prototyping.
- The Spiral approach is most useful in a *research* project, i.e., a project where many options, requirements, and constraints are unknown at the beginning. If the effort is a *development* effort, i.e., a project where the product, options, requirements, and constraints are fairly well-understood, then the Spiral life-cycle is not as appropriate as some other life-cycle approaches.

A Suggestion For The Object-Oriented Life-Cycle: The Recursive/Parallel Model

The approach to object-oriented life-cycles which seems to have the most success is the "Recursive/Parallel Model." Grady Booch, and others, have described this approach as:

- Analyze a little
- Design a little
- Implement a little
- Test a little.

Instead of doing "all of the analysis," followed by "all of the design," the recursive/parallel model suggests that one does analysis where it is appropriate, design where it is appropriate, et cetera.

The recursive/parallel approach was not something which was derived theoretically. It evolved from real people attempting to apply object-oriented thinking to real projects. Anyone who has worked on a project of significant size knows that requirements are not all at the same level of abstraction. Some are broad and high-level, others are very detailed. This means that, once the entirety of the requirements are examined, some requirements may be deferred until a later time.

Likewise, some design, implementation, and testing concerns can be deferred until later. However, to make the recursive/parallel process work, several criteria have to be met:

- The entirety of the information available for a project must be examined to carefully determine which decisions can safely be left until later. Please note, that this is the recommended practice even for traditional life-cycle approaches.
- The interfaces of the system components (i.e., objects, classes, and systems of objects) must be well-defined, and kept fairly constant. (Remember: Walking on water and developing software from a specification are easy if both are frozen.)
- The components of the system must be loosely coupled and highly cohesive.
- Verification and software quality assurance must be part of the overall process.

There are quite a number of issues which must be addressed for the recursive/parallel life-cycle. For example, since this model gives management and the client so much more control over the software development process, how is that handled effectively?

One other observation I should make, and it will confirm a lot of suspicions, is that the differences between analysis and design in an object-oriented project, are far less than in a structured project. Said another way:

Structured methods:

Good News: Analysis and design use vastly different techniques, graphics, and evaluation criteria. It is fairly easy to tell when you are doing analysis, and when you are doing design.

Bad News: The gap between analysis and design is very wide. For example, although there are systematic ways to convert low-level data flow diagrams into structure charts (e.g., transform analysis and transaction analysis), few people seem comfortable with these techniques.

Object-Oriented methods:

Good News: The chasm between analysis and design is very narrow. Overall, object-oriented thinking is much more uniform than structured thinking.

Bad News: It is more difficult to separate "analysis concerns" from "design concerns." The thinking, tools, techniques, and guidelines have much more in common, than they have differences.

Although we will still have life-cycles, and as much rigor (including mathematics) as necessary, in object-oriented life-cycles, we will have to change the way we view the software life-cycle. The good news is that a lot of people have already given this a great deal of thought. Further, the ideas have been tried on a good number of projects, for a few years.

Edward V. Berard *is the founder of Berard Software Engineering, Inc. He may be reached by calling (301) 353-9652 or writing to him at 18620 Mateney Road; Germantown, MD 20874.*

Douglas Graphics Design System (GDS) for highway and bridge design, and geographic information systems applications. GDS is an object-oriented graphic database with products to address computer-aided design, civil and structural engineering, architecture, facilities management, and geographic information systems; as well as extensive database management capabilities.

Source: "McDonnell Douglas Graphics Design System Selected by Departments of Transportation in Maine and Washington," *St. Louis Computing,* December 1989, p. 25.

Analysts who attended the briefing last week at Mentor Graphics' Beaverton, Oregon headquarters last week came away predicting that Mentor's conversion to an object-oriented database will help the company pull further ahead of its rivals. The "Falcon" database incorporates a design management framework, and object-oriented database and an object-oriented programming tool—a combination meant to give engineering organizations a comprehensive set of tools that can be customized for their particular applications and design environment. The switch to an object-oriented environment offers the potential for radical change in the user's ability to customize tools.

Source: "Preview of Release 8.0 'Impressive'," *Electronic Engineering Times,* Loring Wirbel, January 29, 1990, pp. 1, 104.

Giving Motorola one of its biggest design wins yet for the 88000, GEC Plessey has announced that it is rewriting the software that runs on the GEC MarkII processor on the object-oriented C++ derivative of C for the 88000.

Source: "...As GEC Plessey Picks 88000 for Next Generation System X," *Unigram,* February 16, 1990.

Wang began to push its new business package, ClearView, an object-oriented desktop manager. ClearView is designed to run in a SCO Unix environment on the new servers or on the company's new Unix PCs.

Source: "Wang Takes Wraps Off Unix Boxes," Unix Today, February 5, 1990.

Newsworthy Revelations

Object-oriented programming is an ongoing revolution in the craft of software development. But the languages that implement it directly won't take the entire programming world by storm. You won't find megalithic corporations (such as the federal government) throwing out COBOL and FORTRAN for C++ or Smalltalk.

Source: "The OOP Elite," *Computer Language,* Scott Robert Ladd, January 1990, pg.109.

The grim news for the developer who has stayed behind is that user needs and expectations are perhaps moving even faster than the technology. This delay in implementation has led to a crisis in the application development market. There's new competition, [and] backlog... The arrival of networking on the scene has also had some profound implications on application architecture. What you want to arrive at is "adaptable applications" and it's the object-oriented tools that get you there. ...development software into the '90s leads to C, the structured query language for database access, fourth-generation languages, and productivity tools for development—all underlaid with object-oriented databases.

Source: "Applications of the Future," *Directions,* Charles E. Watkins of Texas Instruments, January 1990, pp. 10-13.

The key to programmer productivity is better tools for the programmer. You need tools that allow for rapid development and customization, that make it easy to support a lot of different variations on one product, and that will insulate the program, as much as possible, from the environment in which it runs. And it's the object-oriented tools that will get you there.

Source: "Applications of the Future," *Directions,* Charles E. Watkins of Texas Instruments, January 1990, pp. 10-13.

To give you an early peek at what the new decade might hold, the *Mercury News Business* staff collected their predictions into the following lists. They represent the best guesses of market researchers, venture capitalists, corporate strategists and our reporters. ...Tech that will shine...object-oriented programming.

Source: "The Look of the '90s," *San Jose Mercury News,* January 8, 1990.

As a new decade begins, users can expect to see a continued, gradual proliferation of newer programming capabilities, such as object-oriented programming, fourth-generation languages and computer-aided software engineering (CASE). But, according to several experts, the real power of these new software paradigms will be realized when the most useful mixture of development tools can be integrated into a single software development environment.

Source: "Old Coding Habits Die Hard: Automated Tools Haven't Filled User Need for a Unified Development Environment," *Unix Today,* Mitch Irsfeld, January 8, 1990, pp. 13, 15.

"Of all the industrial expenditures in the world, software maintenance is probably bigger than any other function," [Brian] Boyle [of Novon Research] said. "Companies that think in the long term invest in software maintenance, and the whole thing about object-oriented programming is in the 70 percent of the software money that is spent on maintenance."

Source: "Old Coding Habits Die Hard: Automated Tools Haven't Filled User Need for a Unified Development Environment," *Unix Today,* Mitch Irsfeld, January 8, 1990, pp. 13, 15.

Object-oriented programming offers greater productivity and functionality for both developers and users, but the development community is still struggling with the basic concepts of this new world, and corporations have yet to be shown proof of its potential. ..."The need for doing object-oriented programming isn't readily apparent. The corporate MIS department isn't just going to embrace it, " [John Canestraro, vice president of product strategy for Interactive Images Inc] said. ...Most of corporate America is still a year or two away from seriously considering OOP.

ware manufacturers such as IBM Corp., Apple Computer Inc. and Hewlett-Packard Co. are getting involved. With advanced object-oriented environments such as Office Vision and NewWave, which call for more powerful, memory-packed systems, they may offer the push necessary for object-oriented programming to take off more quickly.

Source: "Richer Environments, Applications," *Computer Reseller News,* Philip Rose, January 29, 1990, pp. 2,29.

Borland International, Inc. broke out of the C++ starting gate last month with the beta-test launch of a new version of C loaded with object-oriented extensions. Because many C Language developers are heralding C++ as the language of the future, the beta release of Turbo C 3.0 appears to give Borland a considerable head start on its most formidable competitor, Microsoft Corp.

Source: "Borland Braves C++ Frontier as Turbo C 3.0 Enters Beta," *PC Week,* Paul M. Sherer, February 5, 1990, pp. 1, 6.

Digital's third major shift in recent years, according to [Samuel] Fuller [vice president of research at DEC], has moved from a primary focus on systems software to software that supports users more directly on an enterprise-wide network: relational DBMS and, more recently, object-oriented DBMS, TP (transaction processing) and a growing array of CASE (computer-aided software engineering) tools.

Source: "DEC R&D Chases a Changing World," *Digital News,* John Cox, January 22, 1990, pp. 1,6.

A new graphical analysis and design product called DECdesign is the first of a series of products to be released under a new product development program, also called DECdesign. DECdesign is integrated with the CDD/Plus repository, and DEC claims it is based on object-oriented technology, allowing it to adapt to support additional current and future modeling techniques.

Source: "DEC's CASE Counterpunch," *CASE Outlook,* Gene Forte, December 1989, pp. 30-33.

Many DEC followers are absolutely convinced that DEC's database will include object-oriented structure imposed over its own RDB relational database. "That's not so," says Debra Curtis, DEC's networked management marketing manager. Curtis says, "the database will be a self-contained object-oriented database with an 'export mechanism' to a relational database." The export mechanism will perform as a function module under DEC's architecture and will translate data from the object-oriented database into RDB-styled data. The simplicity of DEC's object-oriented database will make it easier to change a network. "Relying on object-oriented techniques," Curtis says, "DEC can model anything from a DECnet node to the thermostat in the computer room—and that includes IBM Systems Network Architecture devices."

Source: "DEC's Great Promise Is Not Yet a Product," *Communications Week,* Christine Bonafield, February 5, 1990, pp. 6,10.

Netwise, Inc. of Boulder, Colo., and AT&T's UNIX Software Operation are working together to ensure the interoperability of the Netwise RPC TOOL and AT&T's C++ Language System. The goal is to define a new series of standard object classes making it easier for C++ programmers to develop object-oriented applications for multivendor networks.

Source: Computer Age—Software Industry Report, February 5, 1990.

Hewlett-Packard's [Palo Alto, Calif.] future applications strategy is based on object-oriented technology.

Source: "The Saga of RPC Technologies," *Applied Networks,* January 1990, p.8.

Hewlett-Packard has introduced the User Interface Platform, a flexible software toolkit for creating graphic user interfaces. The toolkit includes an object-oriented C library.

Source: "HP Toolkit Creates Graphic User Interfaces," *IEEE Computer & Graphics & Applications,* January 1990.

The first stage of acceptance of any new idea is over; object-oriented programming cannot be ignored any longer. Hewlett-Packard proposed object-oriented COBOL extensions to ANSI in August. The second stage is also over. It is impossible to say, "It won't work." Software developers are beginning to advertise for C++ instead of C programmers. Of course we can still say, "It won't work here."

Source: "The Object-Oriented Programming Revolution?," *Inside DPMA,* Jim Jinkins, January 1990.

Hewlett-Packard plans to announce this week what company officials call the first object-oriented Unix platform for multivendor network management from a major computer manufacturer. The development platform is the centerpiece of HP's OpenView architecture.

Source: "HP Readies Unix Net-Management Platform," *PC Week,* Paula Musich, January 22, 1990.

Hewlett-Packard is expanding its suite of CASE products with two new C++ object-oriented programming languages for its HP 9000 and Apollo workstation lines.

Source: Computer Age—*Software Industry* Report, February 5, 1990.

Hewlett-Packard, through its Apollo Division, has introduced the HP Text Management library, an object-oriented, portable software toolkit. The new kit is said to enable software developers to produce text-based applications more easily and efficiently than currently is possible with conventional text-manipulation tools.

Source: "Object-oriented Toolkit," *TC Interface,* February 1990.

Kennedy Space Center plans to procure Lisp machines, processors, compilers and software, including several utilities and extensions. Software development tools will include an object-oriented programming system.

Source: "NASA," *Federal Computer Week,* January 29, 1990.

The Maine Department of Transportation has purchased McDonnell

A Data Manipulation Language for C++ Developers—A Proposal

by Thomas Atwood

One of the most important questions to consider as OO Databases emerge is what the database manipulation language (DML) for the OO generation will be. While the goal is obvious —develop a standard that fosters interoperability between software application and DBMS products from different vendors—how to achieve the goal has not been addressed. In my mind, what needs to be done is clear.

We should establish a two-pronged approach to establish DML standards for the object-oriented world.

First: We as an industry should establish a DML language binding appropriate to each language for developers.

Second: The industry should establish an interactive DML that will focus on end-user needs and is language independent.

In this way, we as an industry can adopt the best technology for developers and end-users without making compromises that result in neither group's needs being met. In this article I will focus on why the language binding approach is the right way to establish a DML language binding and in particular what would be done for C++. Before explaining my proposal, let's examine the lessons of the Relational database era.

Lessons from the Relational ERA

"Those who ignore the lessons of history are condemned to repeat them."

While SQL is widely regarded as a standard, it has done little to ensure the interoperability of software among multiple DBMS products. Oracle, Ingress, and their brethren have each extended SQL in ways, that if used, render you captive to their products. Why have customers tolerated this? Good marketing? Maybe, but something more fundamental is at work.

The Relational era defined SQL as the DML for both developers and interactive users. It has been less than adequate for both purposes. From a developer's perspective, SQL was proposed as a standard to work with all languages: C, Cobol etc. Unfortunately SQL often clashes with its host language. It has its own definition of expressions, statements, and variables that conflicts with the definition of expressions, statements, and variables in the base programming language. The developer is then faced with a choice of implementing functionality in SQL or in the host language. Developers either choose the language they are most familiar with, or mix and match, using the host language when SQL proves inadequate. To encourage the use of SQL, vendors extended the SQL sublanguage to improve it. Thus the standard was violated. The lessons is clear:

- Having SQL as a standard was good because it helped foster a new generation of technology, but
- a standard with weaknesses will be compromised, thus sacrificing many of the benefits of having a standard.

What Should Be Done for the Object-Oriented Generation

We as an industry should establish a DML language binding appropriate to each language for developers. The DML syntax should fit smoothly into the host language syntax, and take full advantage of the strengths of the host language as opposed to competing with them as does SQL. A major benefit of this approach is that it would reduce the learning curve for developers.

How This Would Work

Historically, each database generation has moved through three levels of binding a DML to a programming language —bad, better and best.

BAD: Doing the DML through subroutine calls at the source code level is the worst implementation. The basic problem is that it (1) is cumbersome for the programmer, (2) introduces extra syntax that makes it impossible to retain binary compatibility with existing libraries of software, and (3) imposes needless performance overhead.

BETTER: The second level is a preprocessor interface, in which a DML syntax has been introduced, but the preprocessor used to translate it into subroutines and calls to the DBMS runtime is only capable of parsing the DML sublanguage, and therefore has to be told which lines are DML. The problem here is that the programmer cannot use programming language variables in the DML statements, and conversely cannot use database variables within statements or expressions in the base programming language. Not only does this limit the efficiency and expressiveness available to the programmer, but it makes it difficult to ensure the safety and integrity of data with this approach.

BEST: The best approach is a preprocessor capable of handling the full programming language as well as the DML constructs. The preprocessor is used to translate the DML constructs into lower-level programming language statements and calls to the DBMS runtime.

The preprocessor approach allows the programmer to freely intermix variables from the base language and variables from the DML in a smooth, intuitive fashion. This produces smaller, faster, simpler, and easier-to-read programs, which is, after all, the objective. The benefits of this approach are enormous for developer productivity. Unlike SQL, the developer has no barriers

between his program and his data.

The preprocessor would output C++ or C code at the developer's discretion to ensure maximum portability and adherence to the C++ language standard. Interestingly, my recommendation for the preprocessor approach is directly analogous to what has occurred in the Relational era. Today you can buy language compilers that embed their version of SQL into the host language from most of the leading relational companies.

What Should Be Done to Establish a DML for C++?

An ideal opportunity to address these issues is at hand. X3J16 is the ANSI committee to set standards for C++. At the March meeting at AT&T in New Jersey, the process of formalizing a standard for C++ began in earnest. The current thinking, vocalized best by Bjarne Stroustrup of Bell Labs, inventor of C++, is that the C++ language should not be extended to incorporate a DML. I agree. Rather, the data manipulation language should be added as an appendix to the C++ language standard in the same way as Cobol established an appendix for SQL.

What About Interoperability with SQL Databases?

The data model for an object-oriented database is defined to support both encapsulation and inheritance, and is very different from the data model that is assumed by SQL. Attempts to use SQL for OO-DBMS would thus sacrifice many of the benefits of object-oriented technologies. This does not, however, imply that companies must walk away from their investment in SQL and relational databases to use object-oriented DBMS technology. All it means is that SQL is not the solution to interoperability.

The solution is to set standards for the OO-DBMS data model and for DML language bindings. With these in place, it becomes a straightforward task for vendors to map between the two DBMS technologies as appropriate and necessary. In the meantime, vendors should put the machinery in place to build bridges between the environments.

Conclusion

The success of OO-technologies will, in part, depend on its implementors learning from the lessons of the past. Users want open systems that support interoperability and that integrate well into existing commercial environments. Users that are developers want all this plus power and convenience. I, for one, recommend that we give users what they want.

Tom Atwood *is the founder and chairman of Object Design, Inc. If you would like to discuss this topic or other topics related to object-oriented technologies with him, please write to Object Design, Inc., One New England Executive Park, Burlington, MA. 01803, or call him at (617) 270-9797.*

Project Management

Case Studies in Prototyping (Part I)

by Adele Goldberg & Kenny Rubin

"No army can withstand the strength of an idea whose time has come."
—Victor Hugo

The substance of this article is an explanation, through examples, of how prototyping in the software development life cycle can succeed or fail. The case studies that we have selected are actual development efforts undertaken by our colleagues. All were written using object-oriented languages. We have masked the examples to protect the not-so-innocent.

For each of the examples, we provide a description along the following dimensions:

- Project Description
- Team Composition
- Process Setup and Execution
- Outcome

Our purpose in presenting these case studies is to illustrate by example the factors that influence the success or failure of a prototyping effort. The taxonomy of prototypes employed—quick-

Publicity Corner

Clippings Summary

The number of clippings totalled 157; 55 articles in weeklies, 36 in bi-weeklies, 65 in monthlies and one in a daily. About 29% covered object-oriented programming, languages and tools, 25% were object-oriented hype or marketing news, and 17% were product announcements or reviews. The remainder (29%) of the articles included general technical articles, editorials, and graphic applications using object-oriented technology.

Applications

Quatro Pro uses a technology called VROOM—Virtual Real-Time Object-Oriented Memory Manager. This is a...technique that makes use of relatively small (2k to 4k) program overlays called "objects" that are swapped to and from disk as needed. ... maximizing data space and minimizing program code in memory.

Source: "InfoWorld Hits the Heights with Six Spreadsheet Packages that Rule the Number Crunching Hill," *InfoWorld,* John Walkenback, Tracey Capen and John Richey, January 22, 1990, p. 73.

Object-oriented software is a natural fit for this emerging [digital video and audio hardware] market because such software treats pieces of video and voice as discrete items or objects. Object-oriented languages and databases allow for easier manipulation of the images and voices used in interactive presentations. Oglivy & Mather, the big New York advertising agency, has developed a multimedia PC application that lets account executives call up videos of TV ads and images of print ads. The video and audio segments are accompanied by data indicating how well the ads are working. This application was created using an object-oriented language. ...One small company, Applied Optical Media Inc., has created a multimedia training program for truck drivers. Through interactive video that simulates certain driving procedures, the application rates drivers on how well they understand safety standards and essential rules of the road. The application was created using object-oriented software. ...Computer Enhanced Interactive Technology Inc. has developed object-oriented tools that let end users create their own multimedia applications. But such easy-to-use tools are not popular yet, which means that some end users must hire outside consultants to create large-scale applications. ...Andersen Consulting is taking advantage of the need for such services. The firm has created a quality assurance application for Intel video hardware. Andersen application compresses videos of people working on a factory floor onto CD-Rom disks. Videos are played back when bosses evaluate workers' on-the-job performance. An Andersen spokesman says this quality assurance application treats video and other data as objects. ...The link between object-oriented programming and multimedia applications is a strong one because "object-orientation allows you to develop things in chunks," says Christopher Stone, director of the Object Management Group.

Source: "Multimedia Tools: Objects D' Art," *Computer Systems News,* Evan Schwartz, January 29, 1990.

[In this article, James Martin summarizes:] The RAD (Rapid Application Development) life cycle has major advantages over the traditional life cycle. These include the following: ...Reusability: Object-oriented planning and design are incorporated to achieve the maximum degree or reusable data structures, procedures, components, templates, and designs that are used to generate code.

Source: "User/ Analyst Teams Make the Best Systems Developers," *PC Week,* James Martin, January 29, 1990.

Companies Supporting Object-Oriented Technology

Thanks to object-oriented concepts, computer applications of the not-too-distant future will be highly interactive, richly interconnected with each other and with other systems, and vastly more powerful and capable than they are today. As a matter of fact, such applications will probably change our world more profoundly in the future than they have already. Since these powerful environments will require powerful systems, it is no accident that hard-

Calendar

The HOOT calendar presents conferences and meetings that are focused exclusively on object-oriented technology. It should be noted that many other other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to the Editor, HOOT, 8825 SW. Umatilla St., Tualatin, OR 97062

April 9-11, 1990

2nd USENIX Conference on C++—San Francisco, CA. Tutorials,technical sessions. Conference chair: Jim Waldo. Keynote: Adele Goldberg. Contact: USENIX Conference Office, 22672 Lambert St., Suite 613, El Toro, CA 92630, (714) 588-8649.

May 21-24, 1990

SCOOP-East—Wang Institute of Boston University. Co-sponsored by JOOP and BUEC. Tutorials, Panels, Product Presentations and Exhibits. Keynote: Niels Jensen, JPI. Contact: Bob Daniels, BUEC, 72 Tyng Rd., Tyngsboro, MA 01879, (508) 649-9731.

June 6-8, 1990

Eurographics Workshop on Object-Oriented Graphics—Konigswinter, Federal Republic of Germany. Workshops cover standardization and OO languages for graphics. Workshop, CWI, Kruislaan 413, 1098 SJ. Amsterdam, The Netherlands. Tel. 31-20-592-4058.

June 26-29, 1990

TOOLS '90—Second International Conference and Exhibition. CNIT-Paris, France. Program Chair: Jean Bezivin. Conference Chair: Bertrand Meyer. Tutorials, workshops, technical sessions. Contact: TOOLS '90, SOL, 14, rue Jean Rey 75015 Paris, France. Tel. 33-1-40-56 0358.

July 16-18, 1990

SCOOP-Europe—Imperial College, London, UK. Cosponsored by JOOP and BUCEC. Keynote-Adele Goldberg. Tutorials, panels product educ., exhibits contact: Chantal Olsen, BUCEC, 72 Tyng Rd., Tyngsboro, MA 01879, (508) 649-9731.

Sept. 14-15, 1990

Symposium on OOP Emphasizing Practical Applications—Marist College, Poughkeepsie, NY Keynote. Bertrand Meyer. Tutorial Richard Wiener. Technical sessions, contact: James Ten Eyck, Marist College, Poughkeepsie, NY 12601, (914) 471 3240 Ext. 606.

anyone. This is not to belittle the role of the language but to clarify that language is a means to an end; that learning the language is just not enough. Emphasizing the concepts underlying good object-oriented design and gives the student a feeling for how an object oriented language serves the realization of these concepts. The concepts themselves are true for software design in any language. Object oriented languages merely make their application easier. Some of these concepts are listed below:

Design is modelling the world

The world-it is a mapping of the problem domain to software space and is therefore a representational task. The better the problem representation, the more useful and pliable the solution.

Clearly state the problem

Identifying the problem correctly is half the solution—the model practically falls out of a good problem statement.

Think Names— "If it ain't got a good name it don't exist"

Good names for problem domain elements are essential to getting a grip on the problem. Imagine a baseball commentator saying "The guy at an angle of 37 degrees from the base reaches out to stop and hold on to the globular structure made of leather, meanwhile the fella who swung the wooden instrument drops it and makes his way hastily to the circular spot..."

A true enough account of the game, but lacking entirely in good, evocative names that capture what is happening. The real commentator would probably say "The SHORT-STOP reaches out and CATCHes the BALL while the BATTER RUNS to FIRST BASE".

Think Purpose

The same problem can be stated at several levels of complexity. The solution should address the right level. For instance, my representation of the solar system might consist of just the sun and the nine planets executing in circular orbits. An astronomer's model would be much more complex. The point is, there is an adequate level of solution that the programmer should strive for, not the ultimate solution.

Think Client

Who uses the software you write? Who is the Client? Each piece of software has a Client. Keep the client's needs in perspective throughout the design. This puts a realistic upper and lower bound on complexity. Often the client may be unable to express the problem completely. Such ambiguities should not get into the design; the longer they remain there, the more havoc they wreak.

What do your clients see? How will they use your software? Make it as natural and easy for the client as possible. How much will one need to learn to use your software? Impose as little as possible upon the user. Suppose as little as possible about his prior knowledge. If there are suppositions, make them explicit.

Think prototyping

This is where object oriented languages really help because they allow a more natural representation of the problem than other languages. Often, it is as easy to specify the problem in code as in words. The truest description of the problem lies in its working solution! Often, prototyping will give you insight into the problem that you would not otherwise get.

Think Temporary, Think Remodel

Prototype different approaches. Do not be shy of changing early in the design process. There is no perfect representation of Reality, only various approaches to suit different needs. There is nothing sacred about any design—all designs are temporary. Given this, you will filter out transient approaches and preserve what is solid and basic to your design.

Think non-object oriented

Not everything needs to be an object. Don't proliferate objects. Create objects only when essential. For instance, there is no need to create a class in C++ if nobody is going to derive from it or refer to it as an object.

"The strong got stronger, the weak got lost"

That is how Ragu Raghavan of Mentor Graphics Corporation summarized his company's experience in moving to C++. It is important to note that an object oriented language facilitates improved productivity, increased reliability, reduced code size, etc. but does not automatically guarantee those results. On the other hand, an object oriented language without a design mind set is a season ticket to chaos.

The best language is no substitute for that basic ingredient of good software—thought. Object oriented languages can and do serve as better vehicles of thought, but as with most vehicles, their utility is ultimately determined by their cargo.

Niranjan Ramakrishnan *is a training consultant at Semaphore Training. He may be reached at Semaphore Training, Suite 200, 800 Turnpike Street, North Andover, MA 1845; (508) 794-3366.*

and-dirty, mockup, throw-away, exploratory, design-driven—is borrowed graciously from Connell and Shafer (*Structured Rapid Prototyping,* Yourdon Press, Prentice Hall, 1989).

The article is divided into two parts because of the length of the examples. In this Part 1, we present four case studies, each representing either a quick-and-dirty or mockup prototype. Part 2 will appear in next month's issue, covering throw-away, exploratory and design-driven prototypes. The terminology of analysis and design prototypes was introduced in the March issue.

Quick-and-Dirty Prototyping

The philosophy behind quick-and-dirty prototyping is to use the computer as a sounding board for ideas, to find out if you understand and can unambiguously express a concept, technique or approach. The process involves using anything at your disposal to get a running computer result that can serve as a demonstration of your ideas. The trick behind successful quick-and-dirty prototyping is to get the result quickly, make an evaluation, and then move on to the next stage of development, leaving the prototype behind.

Case Study 1. A User Interface Framework

"The cautious seldom err."
—Confucius

"The man who makes no mistakes does not usually make anything."
—Edward J. Phelps, Speech at Mansion House

Project Description. The project goal was to create a new approach to interacting with the complexities of a computer operating system and the administration of that system. The general idea was to create a way to interact with computer-accessible information such that the user would have direct visual and physical means of moving through information structures in order to access a wide variety of content stored within those structures. The project was carried out under contract. Time was short, manpower shorter. So the approach taken was to create a prototype of a framework consisting of an information structure representing an arbitrary directed graph, and techniques for traversing that graph. The particular nature of the presentation graphics (the "look") and the menu-driven command system (the "feel") were handled as separate modules so that it would be possible to experiment. Several combinations were devised and tried out during this analysis phase.

Team Composition. The team had six members.

1. One framer or "carpenter." This team member was a programmer who was interested not in exact details of the ultimate application, but rather in the creation of (a) a reusable structure upon which related kinds of applications could be built, and (b) a set of editors for manipulating or specializing the structure.
2. Two applications programmers. These programmers were assigned the task of developing the end-user system by building onto the more generic framework. One programmer focused on the part for accessing the content of the operating system, and the other programmer handled the part for administering the operating system.
3. Documentor. The documentor joined the team shortly before delivery, to create a user guide. Neither systematic design documentation nor on-line documentation was created (a mistake paid for later).
4. Graphic artist. The user interface graphics were done first as a mockup prototype by a professional artist. The drawings were replicated on the computer to determine the actual effect of different display-screen resolutions.
5. Product manager. This person was nontechnical, and focused on coordinating communication between the development team and the contractor.

Process Setup and Execution. The plan was to quickly create example frameworks and styles of interaction to be tested by target users. The framer and first applications programmer worked in tandem to get several prototypes running before the second applications programmer joined the team to extend the functionality. The project was staged in two phases, so that the difficult task of designing and testing a direct-manipulation editor for dialogues with the operating system could be separated from the basic framework and application development. The first phase was delivered on time—basically as an analysis prototype for end-user testing.

Outcome. The first phase of the product was delivered as required by the contract. Subsequently, five additional and very different applications were tested using the analysis prototype. This work gave clear direction on how to redesign the underlying information structures to produce a more robust framework. The second phase of the contract was not carried out because of a change in market requirement. However, the experience lent credibility to the use of rapid prototyping for proof-of-concept presentations before reaching closure on the product specification and design.

Overall, this was a failed project. The functional specification was incomplete, so that there was no certain way in which to estimate the amount of effort needed to complete the project. Nonetheless, there was a predetermined time for completion. The first prototyping phase yielded an interesting result: an exciting approach to solving the stated problem. Because of the time pressures, however, no time was available for discarding the code of the analysis prototype in order to create a new implementation design based on the now fully-understood specification.

Case Study 2. A CASE Environment

"To stumble twice against the same stone is a proverbial disgrace."
—Cicero

Project Description. Having created an analysis prototype for a user-interface structure, the programming team for this new

project unfortunately grabbed the code from the company's files for reuse in creating a prototype of a new information-analysis environment. The idea was to map the information to be analyzed onto directed graphs and then use the already existing user-interface structure for traversing the information space. The initial goal was to create a proof-of-concept.

Team Composition. The team began as one programmer, out to prove that a new, interesting and useful analysis tool was feasible and would be well-received in the market. Later, a large crew of revisers, testers and documentors was added to bring the result to product delivery.

Process Setup and Execution. The analysis prototype was quickly built out of the code created for the user-interface framework (Case Study 1). In this first demonstration system, all of the desired functionality was implemented in order to make sure that the new ideas were fully understood. The system was shown to potential customers, who provided positive feedback about the concept.

Outcome. Because the prototype was the quick-and-dirty kind, the correct next step would have been to throw away the prototype and redesign the architecture as indicated by the results of the prototyping process. Instead, the prototype was evolved to extend both the amount of information the tool could handle and the functionality. No design documents were created; no design review was held. The programmer, empowered by the development technology and tools made available to him, continued the evolutionary process, flying solo, to create the extended system.

Once the second prototype was completed, it was again shown to customers. Management, excited by the customers' positive reactions, turned a feature-driven project into a date-driven product. Several programmers joined the team to do space and performance tuning, despite concerns about the underlying architecture. Because management failed to control the staging of an analysis and design prototype, neglected to obtain appropriate evaluations at each stage, and focused on a promised delivery date, a prototype became a product. A good quick-and-dirty prototyping project was thus a major failure (disregarding subsequent market acceptance of the product!) because an appropriate foundation for building a product line was not created; costly redesign and reimplementation were necessary to recover from the mistake of delivering a great demo.

There was no company librarian assigned to control access to code for reuse in new projects. This mistake permitted an open access policy to prevail, whereby a throw-away prototype became the ill-suited starting point for a new project.

Mockup Prototyping

The mockup prototype created at the analysis stage shows the potential customer or user what the product will look like, helping an otherwise disbeliever to understand the ultimate intention. But the analysis prototype is the proof-of-concept—it is not the product. What you see is far from proof that a robust product can be built.

The strategy behind mockup prototyping is to quickly create an illusion. As in quick-and-dirty prototyping, the process involves using anything at your disposal to get a computer result that can serve as an initial demonstration of your ideas. However, in this case, the result is supposed to bear true resemblance to the result that you expect to create when you complete the product-building stage.

Some authors assume that a mockup is nonfunctional. In the examples we provide, functional mockups were used to help the target end user get a sense of the interaction as well as the presentation.

Case Study 3. A Decision Support System

"Believe only half of what you see and nothing that you hear."
—Dinah Mulock Craik

Project Description. The objective of this project was to design a proof-of-concept for a decision support system consisting of a variety of novel ways in which to access and examine business-related information. Assuming a large and growing database about business documents and business processes, the basic idea was to present the user with a computer-based information center to retrieve documents, examine processes, and generally ask "what if?" questions. The target user was assumed to be computer illiterate, so ease of use was an important design criterion.

The project was done on speculation. The intention was to present it to a potential client who would, by seeing the demonstration of the prototype, understand the enormous potential of the idea and then contract to have the actual system developed.

Team Composition. Two programmers were allotted three months in which to create the prototype. A documentor was added at the end of the project, to create a user guide that could be given to the target client for exploring the prototype.

Process Setup and Execution. The two programmers were new to the object-oriented development system that they used to create the prototype. However, the system included a large and rich library of reusable objects, notably special visual effects and user-interface widgetry that were developed by other programmers in their company. Some of these capabilities were not yet available commercially and were quite innovative. The two programmers quickly learned how to combine the existing objects and make the required mockup prototype. They created an illusion of the full functionality that they envisioned, but did not implement most of the details.

Outcome. The illusion was sufficient to explain their product concept. In fact, they created a mockup of an "application of the '90s." The target client agreed and the contract was signed. The user guide served as an initial functional specification as the programming team entered the design prototyping phase.

Case Study 4. Virtual Reality/Cyberspace

"Light is the task when many share the toil."
—Homer, Iliad

Project Description. The general goal was to support an alternate user interface consisting of physical devices worn by the users; for example, a data glove or a stereo-vision headset. Using one or more of these devices, the user is given a three-dimensional sense of entering into a computer-based simulated world and interacting with objects in that world. One particular project focused on designing a framework in which it was possible to design different specific cyberspace worlds, each able to get input from a variety of devices. The software created for this project consisted of sophisticated 3D interactive graphics, and tools appropriate for manipulating such graphics. In addition, the system had to allow for the specification and editing of events and objects in the simulated worlds.

Team Composition. The team consisted of one project manager, unfamiliar with object-oriented technology, and five programmers who were interested in, but also inexperienced with, the use of object-oriented languages. The programmers insisted on using an object-oriented language, and the manager decided to accommodate them.

Process Setup and Execution. After one year and about 15,000 lines of code, a satisfactory prototype was completed and demonstrated. People who saw the prototype were impressed, and believed they were seeing the next generation of user interfaces and entertainment. The programmers knew that their prototyping work would require a great deal of user-interface graphics; they had believed that an object-oriented language would make their task easier because they identified graphical user interfaces with object-oriented technology. However, they selected a language for which there did not already exist a large body of reusable graphical objects, so they had to develop these themselves. This meant that the prototyping project took longer than expected and was more difficult than anticipated. In addition, because the manager was unfamiliar with object-oriented technology, he did not understand the difficulties the programmers experienced and could not make recommendations for overcoming the project problems.

Outcome. With more effort than expected, the prototype was a success in the sense that an impressive demonstration was created. This demonstration was the stated objective of the project. However, probably because the manager and programmers did not invest time in learning about object-oriented analysis and design, the software system itself was disappointing in that it was hard to understand and to modify. The team members misunderstood what was necessary to build a successful prototype in that they relied on a language alone.

To Be Continued...

A pattern is forming from these examples indicating how to properly stage a project and build a project team in order to take advantage of the rapid prototyping properties of object-oriented systems. This article will be continued next month, with three more case studies and then a summary of factors contributing to the success or failure of a prototyping project.

Dr. Adele Goldberg *is President and CEO of ParcPlace Systems. Adele spent 14 years as a research and manager at Xerox Palo Alto Research center prior to founding ParcPlace Systems. She has served as president of the ACM and has received the ACM Systems Software Award. She has written extensively in the object-oriented field.She may be reached at ParcPlace Systems, Inc., 1550 Plymouth Street, Mountain View, CA 94043; (415) 691-6700.*

I Already Learned in Kindergarten All I Needed to Know About Object-Oriented Design

by Niranjan Ramakrishnan

The Illusion

In most programming classrooms, one learns a language. Frequently, what passes for programming expertise is actually language expertise—familiarity with syntax. Perhaps this confusion, more than any other factor, is responsible for the inscrutable and unmanageable code infesting so many of our software projects. And now in this already murky world of language wizards and software sorcerers, the stirring of the cauldrons of bit-brew is being accompanied by a new magic chant, "Object Oriented!!!"

Managers look at their employees and say, "They have to get object oriented", rather in the manner of parents speaking of their kids needing cholera shots. "Object oriented" must rank among the one of the world's most misused terms. The myth is spread far and wide of 'object oriented' as a cure for all ills. Amidst this hysteria, what does it mean to teach object oriented programming? Below, I suggest some fundamentals that are crucial.

The Concepts Approach

The first task in teaching object oriented design is to take the mystique out of the term, emphasizing that the rudiments of object oriented design are simple, natural and can be understood by

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes,*** I want to be plugged into the latest thinking and developments in the object-oriented technology. Immediately enter me as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	**2 years (24 issues)**
☐ ~~299~~ $249	☐ ~~$598~~ $498

outside the U.S.—add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT—U.S.currency drawn on U.S. bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card #_____ Exp. date_____
Signature _____

Name _____
Company _____
Address _____
City _____ State_____ Zip_____
Phone_____

Send Subscription Orders To: HOOT, P.O. Box 3000, Dept. HOT, Denville NJ 07834
For immediate service—call the Hotline 212-274-0640 or Fax 212-274-0646

HOTLINE on
Object-Oriented Technology
The International Newsletter

588 Broadway, Suite 604
New York, NY 10012
(212)274-0640

HOTLINE on
Object-Oriented Technology
The International Newsletter

May 1990 **Volume 1, Number 7**

Object-Oriented Development: Not Business as Usual

Mark Lorenz

Object-oriented tools and techniques are the best chance we have to work off our programming backlog, dig ourselves out of the maintenance tarpit, and survive the programmer shortage. There has not, however, been much discussion or resolution of the organizational changes that will be required to support these techniques. This article focuses on the organizational issues of an object-oriented development effort.

The Methodology

In order to understand the implications of object-oriented (O-O) development, we need to understand the differences between O-O and current methodologies. There are two major differences:

- *There is a mental paradigm shift from function-oriented to object-oriented analysis, design, and implementation.*

One of the benefits of an O-O model is that it corresponds directly to the real world. You deal with patients and prescriptions in a hospital system, for example, instead of converting the application into objects that a computer deals with, such as integers and stacks.

The greatest benefit in using an O-O paradigm is the increased support for and probability of software reuse. This is the key to increased productivity. Polymorphism, small method sizes, encapsulation, abstract classes, inheritance, and a supplied set of components all contribute to greater reuse.

- *There is a development methodology process shift from a rigid waterfall approach to an iterative prototyping approach (see Figures 1 and 2, Page 3).*

A key part of an iterative prototyping methodology is having a running prototype during the entire development effort. This allows users to be involved in the system being built for them, resulting in the requirements being verified early on. It also facilitates the proof of the viability of tough technical areas. Finally, the complexity of the system being dealt with at any one time is reduced, since you don't try to understand everything up front. As new pieces are explored, they build upon the existing base.

While O-O development could be pursued using any of the many O-O programming languages available today without changing the development process (see the second point above), this would only achieve a small portion of the benefits available. The remaining discussion will focus on the implications of this process shift.

The (Generic) Organization Today

In today's software development organization, the involvement of interdependent organizations – such as Test, Quality Assurance (QA), Configuration Management (CM), Human Factors, and Documentation Support (DS) – occurs relatively late in the development process (see Figure 3). The organizations perform their tasks as follows:

Development

Write detailed functional requirements from initial user requirements. Review. Write high-level design document. Review. Write detailed design document. Review. Write code. Walk through code. Unit test.

Test

Begin staffing up shortly before the first software components begin completing unit testing. For each unit of work, test cases are written and run. Problems are fed back to development and tests rerun as necessary. Emphasis is on exercising someone else's code to surface any logic or interface errors.

Quality Assurance

As code is written, participate in walkthroughs to ensure that coding standards are followed and requirements are met. Emphasis is on being the customers' watchdog.

Configuration Management

As code passes testing phases (unit, integration, system), control (i.e. lock up) versions of the code, including remote storage of copies of the system. Emphasis is on being able to recover from mistakes and/or calamities and being able to regenerate previous versions of the system.

Human Factors

As parts of the user interface (UI) are finished, examine and test the interface for consistency and ease of use.

continued on page 3 . . .

INSIDE . . .

From the Editor.....2
Databases.....5
Comparing Relational & O-O DBMS
Project Management.....7
Case Studies in Prototyping, Pt. .2
Analysis & Design.....10
Understanding the Recursive/Parallel Life-Cycle
Publicity Corner.....14
Announcements.....17
Product News.....17
Calendar.....19

FROM THE EDITOR/JEFF MCKENNA

In this month's issue of the HOTLINE, the feature article by **Mark Lorenz** discusses organizational issues when object-oriented development is used. Mark proposes a new organization for software development that recognizes the changes that the use of object-oriented technology encourages.

Adele Goldberg and **Kenny Rubin** continue their series for management with a second article focusing on software prototyping. They continue to examine successes and failures in using prototyping and present an analysis of their experience.

HOOT introduces a new columnist, **Robert Martin** of Ontologic, in this month's Object-Oriented Database column. In his initial column, Robert discusses the power of single paradigm programming.

The author of last month's feature, **Ed Berard**, continues his discussion of object-oriented analysis and design issues in the analysis and design column. In his article, he discusses the use and implications of the recursive/parallel approach to development.

The expanded version of the Publicity Corner continues this month. The new format gives you a more complete view of the growing coverage of object-oriented issues in the trade press.

Two developments of particular interest are reported in this month's HOTLINE:

The first is the announcement by Graphael of France regarding the formation of Object Databases, Inc. HOOT reported in December '89 on the folding of Graphael, Inc. and the tentative formation of Object Databases. I am glad to see that the parent company (Graphael of France) has decided to go ahead and provide support and ongoing development for G-Base. Many early adaptors had obtained G-Base, and while they knew the risks of adopting new technology, it was disappointing that they suffered from what was basically a management problem. The re-emergence of G-Base as a viable option in the marketplace expands the alternatives for users.

The second is the joint announcement by **IBM** and **Oasys** on the availability of the Oasys object-oriented products from IBM for IBM workstations. The old adage that when IBM embraces a technology it then becomes legit is still true (although perhaps less so now than in the past!) I, and you if you have been reading HOOT, have seen a dramatic increase in the activity of the major computer companies, both hardware and software companies, involving object-oriented technologies.

IBM has been publicizing its connection to objects more and more over the last few months. The availability of C++ on its PCs, workstations and mainframes; C++ being discussed as an SAA 'official' language; the announcement of the availability of NeXtStep on the new RISC workstations; and marketing releases about the object-oriented nature of the Repository and Office Vision are just some examples of this publicity.

IBM is clearly not the only company riding the object-oriented bandwagon. Digital Equipment Corporation, Data General, Hewlett-Packard, Microsoft, and Borland International are all examples of companies 'joining up.'

So, what does it all mean? It means that objects have arrived. They are still in their infancy, perhaps, but they have arrived. I remember talking to users about objects over five years ago and having to spend all my time just explaining the concepts. Now the questions are how they can be successfully applied and how to get going using them. I find myself relaxing a bit; the pathfinding days are drawing to a close.

Does this mean the work of the pathfinders is over? Not by any means. Their efforts now are to help that large and growing group of folks adopting this technology to succeed with the next of generation products and services. There seems to be plenty of work for everyone.

Jeff McKenna
Editor

Advisory Board
Jim Anderson
Digitalk, Inc.
Tom Atwood
Object Design, Inc.
Grady Booch
Rational
Larry Constantine
Consultant
Cort DeVoe
IBM
Adele Goldberg
ParcPlace Systems
Bertrand Meyer
Interactive Software Engineering
Reed Phillips
Knowledge Systems Corp.
Trygve Reenskaug
Taskon A/S
Bjarne Stroustrup
AT&T Labs
Dave Thomas
Object Technology International
Zack Urlocker
The Whitewater Group
Eugene Wang
Borland International

Richard P. Friedman
Group Publisher
Margherita Monck
General Manager
Susan Culligan
Typography & Design
Melanie Younossi
Marketing Manager
Gregory Tsichlis
Circulation Manager

The Hotline on Object-Oriented Technology (ISSN #1044-4319) is published monthly by SIGS Publications Inc., 588 Broadway, New York, NY 10012 (212)274-0640. ©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher. Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30.
POSTMASTER: Send address changes & subscription orders to HOOT, 588 Broadway, New York, NY 10012
Submit editorial correspondance to Editor, HOOT, 8825 S.W. Umatilla St., Tualatin, OR 97062

CASET Corporation
33751 Connemara Drive
P.O. Box 939
San Juan Capistrano,CA 92693
voice: (714) 496-8670
fax: (714) 661-5463

IBM Markets Oasys Software Development Tools on IBM Workstations

Oasys, Inc. announced a Cooperative Software Agreement with IBM under which IBM will market Oasys' software development tools directly to customers in the U.S. As critical components in IBM's technical CASE strategy, Oasys software tools provide the complete development environment for embedded/cross and object-oriented programming (OOP) project teams. Oasys embedded/cross software development tools (for 680x0, 88000, and 80x86 target CPU's) and Glockenspiel C++™ are available on the IBM RISC System/6000™, the RT System™ and the Personal System/2™, all under AIX™. Oasys will continue to provide customer support directly to all end-user customers.

"The availability of the Oasys tools directly from IBM provides end-users with a set of proven, field-tested tools that boosts productivity of their programming staff immediately. Several CASE developers are using Glockenspiel C++ from Oasys to port their applications to the RISC System/6000. We expect the availability of our cross/embedded development tools will continue to open up new markets for IBM with their comprehensive CASE solutions," says Mike Olfe, General Manager at Oasys.

Oasys offers object-oriented programming (OOP) tools including Glockenspiel C++, a proprietary implementation of AT&T's C++ language, both version 2.0 and version 1.2. Glockenspiel C++ enhances the C language and reduces the development cycle by providing such features as multiple inheritance, optional strong type-checking, extensive data abstraction, classes with scope, overloading of new and arrow operators, and data hiding. Screens++™, another Oasys OOP tool, is a set of class libraries that enables application developers to quickly create user interfaces for UNIX applications.

Oasys also offers a set of Cross Tool Kits for 680x0, 88000, 80386 and DOS target CPU's. For the Motorola 680x0 and 88000, the tool kits include Green Hills Optimizing Compilers (C, Pascal, and FORTRAN), Oasys Assembler/Linker/Librarian, Oasys Simulator, and UDB™, (Universal Remote Debugger). For 80386 cross development, the tool kit includes the Green Hills 80386 Optimizing Compilers (C, Pascal, and FORTRAN) and the Oasys 80386 Assembler/Linker/Librarian. The Oasys Microsoft Cross Development System for DOS and 8086/186/286 targets includes cross versions of the Microsoft C Compiler and Macro Assembler/ Linker. Oasys cross tools allow developers targeting a DOS or embedded environment to take full advantage of the speed of the RISC System/6000.

The RISC System/6000 contains many features that will optimize the performance of this software, including the industry's most advanced floating point processor, new super scalar possessors and optimized 3-D graphics capabilities.

Glockenspiel C++, version 1.2, is available immediately on the IBM RT and RISC System/6000. Glockenspiel C++, version 2.0 which includes multiple inheritance, is available immediately on the RISC System/6000 with prices starting at $1,400. The Oasys Cross Tools for 680x0 and 88000 and target CPU's will be available on the RISC System/6000 in third quarter, 1990, with prices starting at $7,800 for a complete cross/embedded tool kit. The Oasys 680x0 and 88000 Cross Tools are also available now on the PS/2 under AIX and OS/2 for $2,625. The Oasys Microsoft Cross Development System and the Oasys 80386 Cross Tools will be available on the RISC System/6000 in the third quarter of 1990 with prices starting at $3,800.

Oasys offers a complete suite of tools aimed at software development using workstation, mini and mainframe systems running UNIX®, AIX, VMS, Ultrix, and other operating environments. Oasys' products, many developed through OEM, VAR and other exclusive distribution relationships, are available worldwide for native and cross development with full technical support and documentation.

For more information, contact:
Kathleen Pacyma
Oasys, Inc.
230 Second Avenue
Waltham, MA 02154
voice: (617) 890-7889

Calendar

The HOOT calendar presents conferences and meetings that focus exclusively on object-oriented technology. It should be noted that many other other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to the Editor, HOOT, 8825 SW. Umatilla St., Tualatin, OR 97062.

May 21-24, 1990: SCOOP-East – Wang Institute of Boston University. Co-sponsored by *JOOP* and BUCEC. Tutorials, panels, product presentations and exhibits. Keynote: Niels Jensen, JPI. Contact: Bob Daniels, BUCEC, 72 Tyng Rd., Tyngsboro, MA 01879, (508)649-9731.

June 6-8, 1990: Eurographics Workshop on Object-Oriented Graphics – Konigswinter, Federal Republic of Germany. Workshops cover standardization and O-O languages for graphics. Workshop, CWI, Kruislaan 413, 1098 SJ. Amsterdam, The Netherlands. Tel. 31-20-592-4058.

June 26-29, 1990: TOOLS '90 – Second International Conference and Exhibition. CNIT-Paris, France. Program Chair: Jean Bezivin. Conference Chair: Bertrand Meyer. Tutorials, workshops, technical sessions. Contact: TOOLS '90, SOL, 14, rue Jean Rey 75015 Paris, France. Tel. 33-1-40-56-0358.

July 16-18, 1990: SCOOP-Europe – Imperial College, London, UK. Co-sponsored by *JOOP* and BUCEC. Keynote-Adele Goldberg. Tutorials, panels, products and exhibits. Contact: Chantal Olsen, BUCEC, 72 Tyng Rd., Tyngsboro, MA 01879. (508)649-9731.

Sept. 14-15, 1990: Symposium on OOP Emphasizing Practical Applications – Marist College, Poughkeepsie, NY. Keynote: Bertrand Meyer. Tutorial: Richard Wiener. Technical sessions, contact: James Ten Eyck, Marist College, Poughkeepsie, NY 12601. (914)471-3240 Ext. 606.

July 9-11, 1990: Eighth British National Conference on Databases – University of York, UK. Emphasis on O-O databases. Keynote, William Kent, HP (USA). Technical sessions, lectures and exhibits. Contact: Dr. Alan Brown, BNCOD-8, Dept. of Computer Science, University of York, Heslington, York Y01 5DD, UK, 011-44-0904-432746.

Sept. 24-26, 1990: C++ At Work '90 – Wang Institute of Boston University. Sponsored by *The C++ Report* & BUCEC. The International C++ Users Group Conference. Technical sessions, tutorials, panels, BOF & exhibits. Contact Bob Daniels, (508)649-9731.

Deanne Berry
Symantec
10201 Torre Avenue
Cupertino, CA 95014-2132
(408) 725-2759

Lund SIMULA for Sun SPARCstation

Lund Software has released Simula for the high-performance RISC-based workstation from SUN Microsystems. With its 12.5 MIPS, the SPARCstation is currently the fastest Simula workstation. Simula on the SPARC is an attractive choice for customers looking for high performance, and it completes the wide range of Lund Simula workstation implementations: Sun-3, Apollo, VAX/UNIX, VAX/VMS, Data General/AOS. Lund Simula is also available for the Macintosh and ATARI-ST personal computers.

Lund Simula has been available since 1981 and marketed by Lund Software House since 1986. Last year's announcement of a public domain version of Simula for the Macintosh has greatly increased the interest in the language and also in implementation of object-oriented user interfaces.

Simula was the first object-oriented programming language and was also the first OOPL to be standardized. This has resulted in a high level of portability between different Simula implementations, from Lund Software as well as from other vendors. The standard implementations include libraries for a wide range of common problems: mathematical functions, random number generators, text manipulation, standardized file system interface, linked lists and discrete event simulation. In addition, Lund Simula provides interfaces to C, FORTRAN and Pascal, to facilitate use of libraries in these languages and to support easy migration to Simula and object-oriented programming.

Lund Simula contains a high-performance compiler. This makes the system attractive for program development and for education. The source level debugger SIMDEB is a valuable tool for both these purposes. SIMDEB is interactive, and no change in the source code is needed for its use. SIMDEB enables the programmer to set breakpoints, examine and change variable values, follow pointers, display source code and trace program execution.

Simula introduced many of the important concepts associated with object-oriented programming: inheritance, sub-classing, dynamic object generation, strong typing, the notion of qualified references, and virtual binding of procedures. Attribute protection gives the programmer full control over class interfaces. In contrast to many other commercially available object-oriented languages, Simula supports garbage collection, processes (or co-routines), and general nesting of classes and procedures. The stability and portability of Simula is appreciated by its users.

Lund Software will continue to improve its Simula implementation and provide support for additional machine architectures in the future.

For more information, contact:
Lund Software House AB
PO Box 7056
S-220 07 LUND
Sweden
voice: int'l-46-46-134060
fax: int'l-46-46-131021

HOOD Toolset, Hierarchical O-O Design for Ada

CASET Corporation has targeted the rapidly growing market for Ada CASE products by introducing an Ada CASE toolset based on an object-oriented Design Method. The "HOOD" method utilized by the the toolset was originated by the European Space Agency (ESA) for real time requirements. HOOD is becoming the *de facto* standard design method in Europe with adoption and use in projects such as European Fighter Aircraft (EFA).

The HOOD Toolset product encourages the use of hierarchical decomposition to describe objects, hence the name. Object-oriented features of HOOD allow an object, a model of a real world entity, to have both data and operations represented and processed. The HOOD Toolset offers capabilities in requirements analysis, design and Ada code generation.

According to Gary Palmer, Senior Software Engineer, "Writing software in which functions have been specified, that contains few errors, is easy to maintain, and can be reused by other programs is extremely difficult. Most developers use software technology that is tried and time-tested, it works, but it is not as productive as current state-of-the-art techniques."

The goals of HOOD Toolset usage are better, more maintainable software designs and increased productivity of each software engineer, to provide Software Managers with the information they need and the ability to produce software of the highest quality.

As an integrated suite of tools for more than just method automation, the HOOD Toolset consists of specification and design creation tools, interactive graphical and text editors, object management store, and output tools for code and document generation. The HOOD Toolset helps the designer by ensuring consistency between graphical and textural representations of the design, providing a warning of design incorrectness or errors, generating necessary documentation that includes conformance with Department of Defense standards, prompting for input if necessary, generating Ada code, and controlling different versions of the design.

Validation of this tools approach lies in the fact that the HOOD Toolkit was constructed using a Tool Builder Kit and the kit itself is constructed from more basic tool builder kit components or written in Ada or structured C. Rigorous quality mechanisms were required and applied. HOOD has also been built by utilizing a two-tier database. The result offers a "future-proofed" toolset that will evolve in accordance with emerging standards in the international CASE community.

HOOD has been used primarily in medium- to large-scale multi-user Ada software development projects where costs of failure are extremely high. The HOOD Toolkit targets technically complex mission critical projects in aerospace, defense, communications, advanced manufacturing systems, safety-critical applications as well as multi-user and multi-site real-time environments.

HOOD is an effective method for complex real-time systems that yields benefits in the areas of design control, maintainability, and reusability.

The HOOD Toolset is now available on both SUN workstations and Digital VMS/VAXstations running DECwindows. A release in the near future will include additional UNIX platforms running X.11 Windows. Extensive user support and training is available through telephone helpline services and field support engineers. Training is also available for method, toolset and administrator usage.

For more information, contact:
Martin Cody

continued from page 1

Documentation Support

Based on specifications on the UI and functionality, write user documents describing the system.

This organization reflects the process that is geared to work on the traditional, lock-step waterfall methodology. This process was devised to try to effectively manage software development and is in use in most corporations today. Benefits to this approach include:

- *Internal or external legal contracts are easier to write for "accurately" specified pieces of work.*
- *Schedule and resource planning are facilitated.*

The problem is that the waterfall process often results in unsuccessful projects (success being defined as all requirements met on schedule). The reasons this process fails are:

- *In a complex system, it is impossible to understand everything up front in order to specify, design, and construct it perfectly the first time.*
- *Change is inevitable. A process that relies on a static definition and does not plan for change is doomed.*
- *It is difficult to validate requirements when users do not have a medium to allow them to understand what is being built.*

The key point is that schedules and resources need to iteratively adapt to changing requirements and understanding of the system, the same as the system development efforts. This is the reality of what happens, so let's plan for it up front.

A Strawman Organization

I would like to propose a new organization (see Figure 4), set up to support an iterative development process. The new organization consists of the following functional areas:

Class Development

This is a permanent organization to own the component hierarchy. These people will look for and create abstract and reusable object classes in support of the Application Developers. This group could be supplied by people cycling into and out of the Application and Test groups, but needs to consist of people good at cleaning up classes, looking for abstractions, and working with client programmers.

Application Development

This group exists during the lifetime of a particular project. They become (quasi) domain experts *before* trying to implement a solution. They work intimately with the users and true domain experts (they may be the same group).

The Application Developers develop specific applications to solve users' problems, using the available components as much as possible. They submit requests and/or code to the Class Developers for inclusion in the component library.

Test

The Test group works as an intrinsic part of the Application Development group from the start of the project. The leader of the group is a full-time test expert, who manages the process. The other members are supplied by the Application Development group, which cycles people through Test to verify portions of the application that they did not work on.

A library of test cases is kept to be reused. These cases are on-line 'drivers' of the test and include documentation as to their use and what is tested. Ongoing documentation is minimal, with on-line forms to document status as well as a tool that reports results. There won't be time in a rapidly iterating environment for writing a voluminous amount of documentation. The tests will be smaller and more frequent.

More intensive, comprehensive testing at the system level towards the end of development could be used for sell-off, similar to what is done today.

Note that more off-the-shelf, proven components will be used. This is especially important for critical applications such as patient monitoring and nuclear plants. Another function of this group is to look for use of and candidates for reusable components.

Quality Assurance

Quality needs to be integrated into the organization. An independent leader of this function can still act as watchdog. The technical leaders of the Application Development organization need to monitor their coding conventions for certification of code for release.

The QA leader can achieve added traceability in this environment, because the problem domain maps directly to the implementation domain. Tools such as HP's Object-Oriented Systems Analysis (OSA), that facilitate relating words in requirements specifications to objects in the subsequent design and implementation, are already being developed. These allow

continued on page 4 . . .

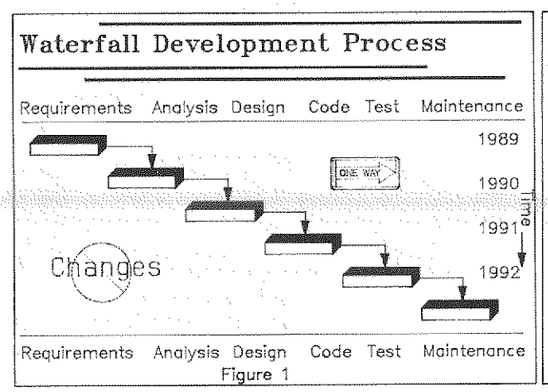


Figure 1

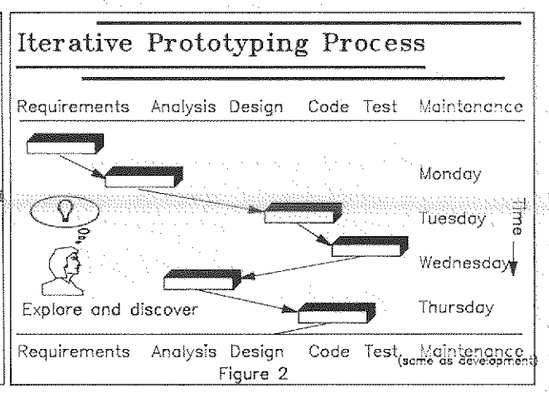


Figure 2

the final product to be sold off when complete (or the problem areas to be identified early).

Configuration Management

There is currently a dearth of tools related to versioning and multiprogrammer development support in the O-O world. This is due to the relative infancy of OODB's and the previous emphasis on single-user stand-alone development. This function should move from a people-intensive effort to a machine-intensive one, through the use of tools.

In an exploratory development environment as we are describing, pieces of software cannot be locked. The emphasis must be on recovery in case of error/catastrophe.

Human Factors

Human Factors can become more involved much earlier in the effort, since there will be user-interface (UI) software almost immediately. The UI can be exercised by the users and feedback given for the next iteration of the system under development. Periodic "releases" of the system need to be planned for this purpose.

Documentation Support

This function should move as much as possible to on-line help. Given the iterative nature of the process, perhaps it is advisable to develop pieces of documentation along with "released" pieces of code. This documentation would have to be maintained along with the code, as it iterates. For this to be realistic, the volume of documentation must be relatively small. Also, a large portion of documentation can be reused, along with related library components.

Developers should be required to make passes at help text when turning over classes for testing and inclusion into the official system build. This implies the need for custom editors. The documentation support personnel would enhance the help for the final product as it is developed. This would occasionally result in work being thrown away, the same as for coding.

Implementing the Organization

A recommended metaprocess for making the organizational move to a new development process is:

- *Gain management support.* Management must support this process and be willing to make the necessary changes for this to work. You will (usually) not be able to get cross-organizational support without the support of upper management.
- *Set up a pilot project.* The best way to gain management support is to demonstrate the payback through a small pilot project. This pilot effort must involve representatives from each functional area (Development, Test, QA, CM, Human Factors, and ID). This will also give you a chance to fine-tune the process.

I've had success in introducing CASE tools that support a Structured Analysis/Design methodology to projects using this technique. This requires support from the people directly involved, who must be willing to try something new.

- *Document the process.* Successful and unsuccessful strategies should be noted.
- *Move the process to the mainstream.* A new project is best for the first major use of the process.
- *Adjust the process iteratively.* Just as with coding, you won't get the process right the first time. Keep the process as a flexible framework that adjusts to changing conditions and information.

Conclusions and Recommendations

When something doesn't work well, you try another way. Our present software development process does not produce a high percentage of successful projects across the industry. An iterative prototyping methodology, supported by O-O tools and techniques, provides a better alternative. The possible benefits include working off your backlog while delivering systems that truly satisfy user needs. The gains require some changes to the way you do business.

There will be organizational discomfort while trying to adapt to a new process. I have tried to give some guidance for a way to start. This will evolve over time, the same as the software being developed evolves into the final desired result. Each organization may find a slightly different variation that works best.

Mark Lorenz is a 13-year veteran of software development, having worked on a variety of software and hardware platforms. He has extensive experience using CASE tools and different methodologies. He has concentrated on the O-O field, especially enabling tools and techniques, for the last year and a half. He lives in Cary, NC.

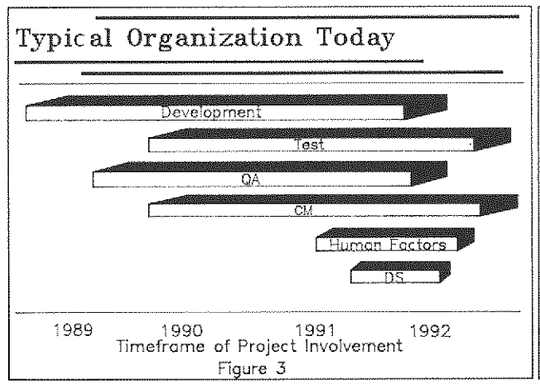


Figure 3

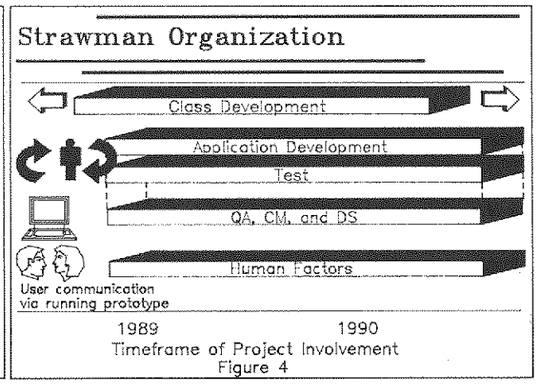


Figure 4

Announcements

Object Databases, Inc., Established in Cambridge

Graphael of France has established Object Databases (ODB) in Cambridge, Massachusetts, to co-develop and market Object-Oriented database management systems (OODBMS).

ODB is starting with a large installed base of object-oriented database users, those that are using G-Base. G-Base users include major U.S. aerospace companies, Federal Government contractors, and other organizations interested in using Information Technology for competitive advantage.

G-Base combines a powerful conceptual data modeling facility based on "frames," with programming capabilities known as "methods." An application's or institution's data model is mapped more closely to the real world than with relational databases where everything must be reduced to tables. At the same time, programming is done in a more modular fashion, allowing for faster development and modification.

In addition to G-Base, ODB offers interactive, graphically-oriented tools to build and query databases. ODB offers a powerful, logic-based query facility, Pro-Query and an expert system building language, G-Logis.

OBD has a new version of G-Base, G-Base/SQL, that supports interaction with relational DBMSs. For those interested in an evolutionary move from relational to object-oriented database technology, G-Base/SQL provides the ability to store G-Base objects in relational tables. G-Base can be used to model the data structures and build the application and at the same time store all data in the relational database.

ODB also offers training and consulting services to help clients succeed in their initial forays with OODB technology.

For more information contact:
William Frank
VP, Marketing and Sales
Object Databases, Inc.
238 Broadway
Cambridge, MA 02139
voice: (617) 354-4220

Product News

Symantec Expands Object-Oriented Support in New Version of THINK Pascal

Symantec Corporation (NASDAQ: SYMC) announced THINK Pascal™ Version 3.0, which includes Symantec's THINK Class Library and support for MacApp® (the class library from Apple Computer, Inc.). A unique class browser has also been added to THINK Pascal Version 3.0. These new features enable THINK Pascal to provide users with the most comprehensive support for object-oriented programming (OOP) of any Pascal environment for the Macintosh available today.

"As a leader in the Macintosh language market, we recognize the demand for a product that eases the transition to object-oriented programming," said Gordon E. Eubanks, Jr., president and CEO. "THINK Pascal's sophisticated class library and tools such as the class browser are critical to making OOP more accessible to programmers."

Features Support OOP

THINK Pascal (formerly called THINK's Lightspeed Pascal) includes the THINK Class Library, which provides code for all the standard Macintosh user interface components and behaviors, such as windows and scrolling, eliminating the need for developers to recreate code. Symantec is the only vendor that provides a comprehensive solution to object-oriented programming for both the Pascal and C (THINK C™) languages, so programmers in either language can reap the benefits of object-oriented programming.

The unique class browser included in THINK Pascal Version 3.0 enables programmers to examine and edit classes and more easily navigate through a large object-oriented program. In addition, debugging capabilities have been enhanced to work more efficiently with object-oriented code.

Apple's MacApp provides an even broader library of objects for developing bulletproof applications. It's a winning combination: THINK Pascal's ease of use and rapid turnaround times and MacApp's second-generation object-oriented framework," said Larry Tesler, Apple Computer, Inc.'s Vice President of Advanced Technology. Tesler, who led the design teams for Object Pascal and MacApp, added that "Macintosh software developers can now create applications with unprecedented ease and speed. My group is already using the product to improve productivity."

With THINK Pascal Version 3.0, programmers have a choice between procedural programming and OOP. The enhancements in Version 3.0 assist developers in moving from procedural programming OOP.

THINK Pascal Version 3.0 also provides improved resource editing, including standalone resource tools licensed from Apple (Rez, DeRez and PostRez). Better segmentation control has been added, along with Projector awareness, improved compatibility with Apple's Macintosh Programmer's Workshop™, and the ability to handle larger source files. The menu structure has been reorganized in THINK Pascal Version 3.0 to take advantage of standard Macintosh conventions. A new "Arrange" command allows programmers to select window layout, and a more flexible Step command offers additional debugging options.

THINK Pascal is used at all levels in the Macintosh programming community. It provides superior-quality code generation and has been used to develop many commercial Macintosh applications, including SoftView's MacinTax; Timeslips III, by North Edge; NuEquation's NuPaint; Serius; and Ceres Software's inspiration 2.0. THINK Pascal is an educational programming standard at major universities, including Harvard, the Massachusetts Institute of Technology, and Stanford University.

THINK Pascal Version 3.0 runs on a Macintosh Plus, SE, SE/30, Portable, II, IIcx, IIcx, and IIcI with one megabyte of RAM. A hard disk and two megabytes of RAM are required for the THINK Class Library, and four megabytes are required for MacApp.

Pricing and Availability

THINK Pascal Version 3.0 is available now. The suggested retail price is $249. Registered users can upgrade to Version 3.0 for $69..

Symantec, founded in 1982, is a leading international software company which provides award-winning application and system software products for IBM PC and Apple Macintosh computer users.

For more information, contact:

attack the problems that decrease productivity.

Source: "This Thing Called OOP...," **DirecTIons**, *Dane Meyer, Feb., 1990, pp. 22-27.*

Software consultants say that one-third of all system-development projects never make it into productive use. "We have to find a way to write software faster and cheaper," concludes George Gardner, a computer consultant with Arthur D. Little Inc. The answer in the years ahead may be a new way of programming computers called object-oriented programming, or OOP.

Source: "Object-Oriented Programming: Untangling the Software Mess," **Industry Week**, *Neal E. Boudette, Mar. 5, 1990, pp. 49, 52, 54.*

"To create a software environment that will match the capabilities of the hardware environment of the future, we will have to do things differently," [Gordon] Bridge [president of AT&T Computer Systems said]. ..."Productivity tools such as powerful, object-oriented languages like the C++ programming language will be vital to business applications requiring fast and simple capabilities for sharing information."

Source: "Software Development Not Keeping Pace With Hardware, Industry Executive Says," **Education Computer News**, *Mar. 6, 1990.*

Companies Supporting Object-Oriented Technology

Ashton-Tate has begun dropping hints about new application development tools based on object-oriented tools and SQL. This would be in line with current trends, but will upset die hard dBASE programmers who don't want to switch to SQL.

Source: **Data Base Review**, *Feb., 1990.*

For its database engine, Ashton-Tate plans to add object-orientation, deductive capability, multi-media support (voice, graphics, and hypermedia), and distributed capability. To the application development environment, Ashton-Tate is adding an integrated dBASE/SQL/object-oriented language with commands to manipulate the graphical user interface.

Source: "What's Going On," **Data Based Advisor**, *David M. Kalman, Feb., 1990.*

Hewlett Packard's New Wave is an object-oriented user interface that includes a special filing system called the Object Management Facility (OMF). ...As part of its DAA, or Distributed Applications Architecture, Data General has stated that New Wave will be one of the supported operating environments of the future.

Source: "CE-oooooh!," **Focus**, *Doug Kaye, Feb., 1990, pp. 44,46-48.*

DEC falls short in delivering software vendors and users a modern applications development platform because All-in-1 [server and workstation software] lacks object-oriented programming capabilities. Gene Hodges, group manager for office information systems at DEC, said object-oriented capabilities are important and predicted they would eventually be incorporated into All-in-1.

Source: "DEC Still Has Some Work to Do," **Communications Week**, *Paul Korzeniowski, Feb. 5, 1990.*

IBM is betting that advanced software supplied by Steve Jobs' Next Inc. will give the company an edge over competitors in the crowded Unix workstation market. ...IBM...will offer NextStep, a software development and graphical user interface environment, as an option for its high-end PS/2s and its new RS6000 Unix workstations. ...[Judith] Hurwitz [analyst with Patricia Seybold's Office Computing Group] said the object-oriented features of the NextStep software will give users an easy and quick way to develop advanced software such as customized document publishing applications.

Source: "IBM Hedges Bets," **Computer Systems News**, *Evan Schwartz, Feb. 12, 1990.*

A controversial article in *The New York Times* on Sunday, Dec. 10, 1989 Entitled *"A Prescription for a Troubled IBM"* by John Markoff, it suggested that IBM rid itself of its AS/400 line of minicomputers and break its ties with Microsoft and Presentation Manager-based strategy. Instead, IBM should adopt Steve Jobs' NextStep, the article suggested.

Currently the operating system of Jobs' struggling Unix-based NeXT machine, NextStep is known for its graphical user interface and object-oriented programming front-end. Most analysts agree that Markoff's idea would do IBM more harm than good.

"Dropping Presentation Manager for NextStep is technically extremely difficult," says industry analyst Esther Dyson. **PC Week** columnist William Zachmann derides the article's suggestions as being "fatal" for IBM.

Source: "Is Microsoft the IBM of the '90s?," **Marketing Computers**, *Alan Brody, Feb., 1990.*

Microsoft sees several major areas in which object orientation will play a major part in products over the next several years: languages and development tools, standard classes and operating system support. ...Microsoft QuickPascal Compiler represents Microsoft's first step into the world of object-oriented programming.

Source: "Object-Oriented Programming: Its Role in Computing," **Library Software**, *Kathryn Hinsch, Marty Taucher, Pam Edstrom , Collins Hemingway, Feb., 1990, pp. 18-24.*

NCR Corp. last week unveiled an application development architecture. ...The key component of OCCA [Open Cooperative Computing Architecture] is Cooperation, an application development tool kit. ...The tool kit includes NCR's System Software Platform, which consists of NCR Desktop – an object-oriented graphical user interface – and a set of industry-standard application program interfaces (API) that enable developers to build communications protocols and other software routines into OCCA-compliant applications.

Source: "NCR Architects Net Applications Strategy, " **Network World**, *Jim Brown, Feb. 19, 1990.*

Standards

The Object Management Group Inc. (OMG) is one step closer to establishing a universal object-oriented applications environment as it puts the finishing touches on a document that fleshes out its goal. That document, which is referred to by the group as the reference model, is expected to be finished next month, according to OMG leaders.

...areas identified in the draft reference model are as follows: a standard structure for object-oriented applications; an object manager paradigm... common services that apply across applications; system services that outline network and operating system standards; and user-interface guidelines.

Source: "OMG Takes Step Toward Object Standard," **PC Week**, *Paul M. Sherer, Feb. 19, 1990*

Databases/Robert Martin

Comparing Relational and Object Database Management Systems:
Single Paradigm Programming is the Key Difference

Previous database columns have introduced object database systems and discussed standardization issues. With this column, we will begin a comparison between object and relational database management systems, starting with the object database concept called single paradigm programming. Object databases have a single, uniform paradigm used for design, programming, and database access. This uniform paradigm is the key factor in providing new levels of performance, productivity, and functionality that set object databases apart from relational systems. This article focuses on single paradigm programming.

The Need for a New Database Model

During the past 10 years, relational databases have moved from interesting theory to dominant data-base technology. This happened because relational systems have many key advantages over files and earlier database systems that make it far easier to build data-intensive applications. Relational benefits include:

- *traditional database advantages of built-in, multi-user concurrency control, integrity and recovery.*
- *a simple, understandable data model.*
- *data independence, which lets administrators reorganize the database without changing applications.*
- *SQL, an industry-standard query language that permits uniform database access and data interchange.*
- *integrated fourth-generation languages (4GLs), making it easy to build forms, reports, and simple applications.*

In spite of these advantages, over 90% of today's applications still use files or older database systems. This is partially due to inertia, as it is often impractical to rewrite existing applications. But the key reason why applications continue to use files today is that they offer the flexibility and performance required to manage complex data. Relational systems do not have this capability, and the slowdown in their growth indicates that many applications simply cannot use them.

Performance, Flexibility, Productivity

Object databases offer a new approach, which couples the performance and flexibility of files with the productivity advantages of relational databases. For applications with complex data, object databases offer file-level performance, which is from 10 to 1000 times faster that relational systems. Unlike files, object databases offer all the traditional database advantages, including sharing, integrity, query, and data independence. Most important, they have single paradigm programming, the key mechanism that delivers new levels of performance, flexibility and productivity.

Single paradigm programming

Like many good ideas, single paradigm programming is a simple, yet powerful, concept. In this approach, a single, simple, uniform model is used for all phases of system development, including analysis, design, coding, and data management. Language and database tools directly implement this paradigm.

Object databases use the object-oriented paradigm, which is rapidly gaining support in the programming community because of its improvements in productivity. They extend this concept to data-intensive applications by tightly integrating object programming languages with database support.

This contrasts markedly with the relational approach, in which separate, disparate paradigms, languages and tools are used in constructing applications. This creates barriers between program design, coding and database access. Crossing these barriers is costly in terms of performance and productivity.

To illustrate this, consider the first program in which you built data structures. It was probably programmed in a modern language such as Pascal or C, languages that have a rich notion of data structures. After you learned the basic concepts of structures such as arrays, lists and trees, you implemented them with the language. Going from basic design concepts to implementation was easy because the language directly supported the notion of data structures.

Then you needed to save these structures on a disk. You were surprised to find out that the programming language couldn't handle this directly. To your chagrin, you discovered that the only way to save things on the disk was to use files or databases. After reviewing the file manuals, you discovered that files do not directly support the same kind of data structures as the programming language. So you had to translate from the programming-language model to the file model. This involved "flattening," or translating structures from the programming-language model to the file-system model. Over one-third of the code in typical file-based applications is required to do this translation.

So perhaps you went looking for a relational database to see if it made things easier. You discovered that databases don't solve this problem, either. With relational systems, you build data structures in C or Pascal, but you program the database with SQL, which requires that all data structures be implemented with records and tables.

If they were tabular to begin with, this was easy, but most data structures are more complex than just a table. To take advantage of data-base functionality, you learned to translate to tables, a process called normalization; but in spite of its fancy name, it still meant you were converting structures from one model into another.

continued on page 6 . . .

What you really wanted was for the compiler to do the work for you. Ideally, that meant developing a concept, and expressing it directly in the programming language. The compiler would take care of managing it in memory and of storing it in a database. This is what single paradigm programming is all about. A high-level modeling paradigm is directly implemented in a programming language that can transparently store information in the database.

Having a single language that can handle in-memory data structures and database access markedly improves productivity. It eliminates the need to write file or database access software. It makes the application software much easier to understand and maintain by encouraging a natural form of modularity. Programs are organized around their fundamental concepts, not along lines determined by the programming tools. The single data model eliminates the need to translate from in-memory forms to on-disk forms, vastly improving performance.

The Power of Single Paradigm Programming

The power of this approach can be illustrated with a simple example. Suppose you have an array, and want to save it in a database. Arrays are well-understood, mathematically sound data structures. Most programs use them. Yet there is no direct way to store an array in the relational model.

You have to translate it into a table. Here is what it looks like compared to an object system ☛

Several points are worth noting:

The object database implements the array concept directly.

Arrays are common data structures that are easily defined in most languages. Object databases provide arrays in a C++ class library. They are declared and manipulated as if they were in-memory arrays in C++.

In relational systems, every data structure must be mapped into a table by the programmer. For the array, there would be a table called ArrayOfInteger with three columns, the array name, the index, and the actual value. In the relational model, all integer arrays would be placed in the same table. This requires the column "Name" to distinguish among various arrays. The "Index" column is also required because rows in a relational table have no implied ordering. These columns are artifacts of the relational model that take up extra space on the disk and in memory. Extra space requires additional CPU cycles to copy and transfer.

Object Database (ONTOS)	Relational Database
Definition:	
int AnArray[10];	create table ArrayOfInteger (Name: char(20), Index: integer, Value: integer);
	insert into ArrayOfInteger values('AnArray',1,0);
	insert into ArrayOfInteger values('AnArray',2,0);
	.
	.
	insert into ArrayOfInteger values('AnArray',10,0);
Access:	
Y:= AnArray[4]	select Value into Y from ArrayOfInteger where Name = "AnArray" and Index = 4;

The object database array is much faster.

The representation of the object database array is highly efficient. In this case, it would contain storage for the integers, which are packed sequentially in a structure. Once an array is in memory, access to its elements takes only a few instructions.

To access the array element, the SQL processor has to look through both the Name and Index columns. To accelerate this look-up, the database administrator would probably define a key for both the name and index. This makes the look-up faster, but slows down writes, and increases database size. In any case, looking up the array is a complex affair and takes much longer than the simple access in the object database.

The object database array is completely safe.

The object database array is type safe. Its implementation has a formal specification that defines how arrays are to be used. The compiler and runtime system enforce this specification, making sure that the proper boundary and argument checking are performed. The SQL shown here would not catch even simple programming errors unless each programmer explicitly checked for these conditions every time the query was used.

The object database array is simple.

The object database code is simple, intuitive and well-structured. All details of the array implementation are hidden from the programmer. This leads to higher productivity. At best, SQL is wordy and, for more complex data types, is difficult to write, hard to understand, and nearly impossible to modify and maintain.

The array is a simple example. For more complex structures, the comparison becomes more one-sided. Many commonly used data structures, including arrays, linked lists, and trees cannot be directly represented as tables. These data structures are well defined and have a firm, mathematical basis. They are used extensively in a wide variety of applications. Yet the relational model does not directly support them, requiring expensive, awkward translation.

Rating Your Application

Single paradigm programming lets you build high-performance applications that are well-structured and easy to maintain. Many people ask us if their applications would benefit from this approach. Our reply is to consider an object database if any of the following are true:

- You have tried relational systems, but found them too slow or complex for your application.
- You are programming in C or C++ and need to save and share data.
- You are using relational databases and have more than five tables.
- You are writing more than 5,000 lines of code and wish to have a well-structured, maintainable application.
- Your application requires non-record data structures such as lists, bags, sets, trees, bit-maps, graphics, video or audio.

In upcoming columns, we will examine other attributes of object data-base systems, including composite object support, flexibility, and extensibility and demonstrate how these attributes provide additional benefits over relational database systems.

Robert Martin, a founder and Vice President of Products for Ontologic, Inc. (the leading supplier of second-generation object database systems), has led the commercial introduction of object database technology. For more information, contact Ontologic in Burlington, Mass., at (617) 272-7110.

Newsworthy Revelations

Object-oriented interfaces – more advanced renditions of the interface first made familiar through the Apple Macintosh –...will flourish in the '90s, so that the "individual will be able to create and use information objects that can be tied logically to data that reposes in other forms on the database," [said Myron Kerstetter, vice president with the Gartner Group]. ...all together...we "effectively allow the use of workstations or the combination of many workstations to work a lot like a network of dumb terminals would [work] when connected to a central processor," Kerstetter explains.

Source: "Workstations Poised to Fill New Financial Niches,"
Wall Street Computer Review,
Victor Kulkosky, Jan., 1990, pp. 25,28-29,32-33,36-40.

...A factor keeping object-oriented programming from taking the world by storm is a reluctance to reinvent the wheel. As I [Scott Robert Ladd of Computer Language] mentioned last month, it is easy for those of us who carry the object-oriented banner to forget we're in a real world with practical problems to solve. Few developers can afford to rewrite existing code to take advantage of object-oriented languages.

Source: "OOPing in Public,"
Computer Language, *Scott Robert Ladd, Feb., 1990, pp. 127-130.*

Users shouldn't think of reusable technologies such as object-oriented programming as a magic that will solve their software problems. Planning and hard work are required to make these technologies pay off.

Sources: **Data Processing Digest**, *Feb., 1990.*
"Cultural Barriers Slow Reusability,"
Datamation, *Nov. 15, 1989.*

While object-oriented techniques have plenty to offer in terms of creating well organized, scoped, maintainable software, there are some disadvantages. Depending on the object-oriented language you choose, runtime performance can be excessive. There are several things that can cause a performance hit...

The formal methodology for object-oriented design is also considered immature when compared to structured design methodologies currently in use.

Source: "This Thing Called OOP...,"
DirecTIons, *Dane Meyer, Feb., 1990, pp. 22-27.*

Object-oriented programming will continue to be a hot topic throughout 1990. Many programmers who know nothing of expert systems will get excited about using C++, Object-oriented CASE tools and Object-oriented databases. (By the middle of the decade, C++ will be the dominant OOP language and will begin to replace C.)

...By 1992 or 1993, the best of the OOP tools will incorporate some inferencing and will be positioned as CASE tools. Some of them will probably be developed by the same vendors who are now selling expert systems tools.

Source: "Predictions: What Will Happen In 1990?,"
Expert Systems Strategies,
Feb., 1990, pp. 11-14.

[The article begins as a dream the author is having about several 'object-oriented' vendors who have just come to town to sell their products, each describing their own as better than the others.]

All those hucksters [vendors] suffered the same fault. They weren't as interested in providing object-oriented tools for computer users as they were in selling products. So they redefined object-oriented to fit whatever they were selling. That's exactly what's going on in the real world. Computer software companies use the term "object-oriented" like food companies use "oat bran." Everything these days has oat bran in it.

...Adding oat bran to a food does not necessarily make it healthier, just as calling a product "object-oriented" doesn't make it any more capable or easier to use. ...Vendors who slap the object-oriented label on their products without actually adding any object-oriented capabilities are lying to you. ...three hucksters were arguing about which programming language was truly object-oriented.

...The fact that one language may have a feature the others don't may make it a better programming language – but it doesn't make it more object-oriented.

Source: "Painted Lemons, "
Computer Language, *Scott Robert Ladd, Mar., 1990, pp. 117-119.*

Object-oriented DBMS purists [design database systems from the bottom up to store data in objects] consider RDBMS approaches [the non-purists who begin with an RDBMS and then integrate object-management capabilities] incomplete, but they welcome the attention. "We thought Oracle, Ingress and Sybase might pooh-pooh objects, but they've validated the market by embracing them," says Tom Atwood, president of Object Design.

Source: "Next Wave Could Be An Easier Ride,"
ComputerWorld, *Ted Rybeck, Mar. 5, 1990, pp. 67-69.*

The drawback is that an application created using OOPS is slower than an identical application written in a conventional computer language. ...But, more important, many an end user is willing to sacrifice performance in order to get software that is easy to update and modify. ...Sacrificing the running speed of a computer program to get more speed out of a programming department makes perfect sense these days. Hardware is cheap. Programmers are expensive.

Source: "Why dirty your hands with programming code when you can just wave your hands and get the computer to do what you want? that's sort of what 'object-oriented' software is all about,"
Forbes, *Julie Pitta, Mar. 19, 1990, pp. 162, 164.*

Increasing Programmer Productivity

Labor costs continue to rise and the difficulty of delivering software correctly and on time continues. ...Statistics indicate that 47 percent of the software that the government orders and pays for is never delivered because the project is either dropped or can't be completed with the funds allocated. Almost 30 percent of government-ordered software is delivered, but can't be used because it is already obsolete. Twenty percent of government-ordered software is abandoned or has to be redone because it doesn't fit the original need. And only two percent of the software the government does receive can be used as delivered.

...Increasing programmer productivity is the key. Some of the methods by which this can be accomplished are: improving methodology, use of better tools, speeding up the hardware, using powerful development environments, moving to higher-level languages, and reusing existing software. ...object-oriented programming has been gaining credence as a way to open up the programming bottleneck. Object-oriented concepts address many of the above solutions and provide mechanisms which

continued on page 16 . . .

Publicity Corner

Clippings Summary

The number of clippings totalled 121: 29 articles in the weeklies, 31 in the bi-weeklies, 58 in the monthlies, and 3 in the dailies.

About 14% of the articles covered object-oriented programming, languages and tools; 19% were object-oriented hype or marketing news, 41% were product announcements or reviews, and 12% were general technical articles. The remainder (14%) were made up of editorials and graphic applications using object-oriented technology.

Applications

Users have stayed away from WISC [Writable Instruction Set Computer] because of the complexity of microcode programming. To solve this problem, International Meta Systems has applied object-oriented techniques to facilitate microcode programming, creating a development environment in which microcode programming is no more difficult than assembly-language programming.

Source: "After CISC & RISC Get Ready for WISC!,"
Computer Technology Review,
Paul D. Ray, Jan., 1990.

Object-oriented programming systems (OOPS) [are] a way to solve development backlogs through reuse and other techniques. Apple Computer and Boeing Co. have experimented with the new discipline. Reported productivity gains of as much as 60% for ongoing projects have encouraged such experiments.

...Tektronix succeeded in getting some OOPS-developed products out but ran into roadblocks in its attempt to use OOPS for anything more than quick prototyping. It is just now beginning to find solutions for some of the early problems it encountered in attempting to introduce object-oriented technologies into its large development organization for production work. Tek found early versions of Smalltalk-80 inadequate for supporting large, multi-developer projects.

So the company switched to AT&T's C++ development language. But it lacked a comprehensive approach to retraining procedural language programmers to use the object-oriented C++. The company never established a set of design and analysis tools specifically for object-oriented designs or the development of consistent objects with the potential for reuse.

Sources: **Data Processing Digest,** *Feb., 1990.*
"Cultural Barriers Slow Reusability," **Datamation,** *Nov. 15, 1989.*

Deluxe Video [program for the Amiga Computer by Electronic Arts] uses a timeline-based object-oriented set of hierarchical events to define a script. A variety of tracks that control events can be placed in a script, and each track can contain a mixture of events and effects, called Scenes. Each option is presented in context-sensitive pop-up requesters. The program is far easier to use than describe.

Source: "Deluxe Video III: New Enhancements for a Great Product,"
MicroTimes, *Matthew Leeds, Feb. 19, 1990, pp. 124.*

Athena Muse [is] an object-oriented system that provides information-processing support for text, graphics, video, and audio. In developing Muse...we have synthesized a multi-paradigm document system that lets users describe the dynamics and presentation of multimedia information without having to resort to traditional programming languages. ...Over the last year, Muse has been used to create many different kinds of multimedia applications, several of which exhibit the dynamic cross-referencing behavior found in hypermedia and other "low structure" systems.

Source: "Musings on Multimedia,"
Unix Review,
Matthew Hodges, Russell Sasnett, Judson Harward, Feb., 1990, pp. 83-87.

The Putnam Companies, Inc. last week said it has built an object-oriented development tool that will enable its programmers to construct distributed network applications with ease. Putnam's Financial Workbench is based in part on the Network Computing System (NCS), a distributed computing technology.

...The tool enables programmers to write applications that spread processing chores among multiple computers on a local area network. The Financial Workbench is expected to give the company a competitive boost by cutting development time for powerful financial analysis applications.

...The company has already used Financial Workbench to develop an application that enables Putnam's financial analysts and portfolio managers to access a wealth of information from data bases on different network-attached devices.

Source: "Distributed Application Tool Pays Off For Putnam,"
Network World, *Jim Brown, Feb. 5, 1990, pp. 1,106.*

By themselves, ...game programs are simply for amusement. For programmers, however, games are often the best teachers of [programming] technique. *[The article discusses how games such as video poker can be a good method for learning about object-oriented programming.]*

Source: "Learn Object-Oriented Programming While You Play Video Poker,"
PC Magazine, *Neil Rubenking, Feb. 27, 1990, pp. 293-315.*

ECSRL [Electronic Combat Simulation Research Laboratory] engineers ...use parallel processing in combination with object-oriented software to approximate real-world physical objects such as enemy ground-based radar or a U.S. aircraft.

Source: "AF Lab Tests AI, Graphical Interfaces for EW Simulation,"
Federal Computer Week,
Mar. 5, 1990.

Among the users developing OOPS applications are American Airlines, Boeing and General Dynamics. Mitchell Oliver is part of a General Dynamics team designing software that will simulate a rocket. Oliver's group is using object-oriented programming tools from ParcPlace Systems. "We had to build a user interface that would allow non-computer literates to be able to use the system," Oliver explains.

Source: "Why dirty your hands with programming code when you can just wave your hands and get the computer to do what you want? that's sort of what 'object-oriented' software is all about,"
Forbes, *Julie Pitta, Mar. 19, 1990, pp. 162, 164.*

Project Management/
Adele Goldberg and Kenny Rubin

Talking to Project Managers — Case Studies in Prototyping, Part 2

"We judge ourselves by what we feel capable of doing, while others judge us by what we have already done."
—Longfellow, Kavanagh

This is the second part of an article that serves as an explanation, through examples, of how prototyping in the software development life cycle can succeed or fail. Part 1 appeared in last month's issue.

The case studies that we have selected are actual development efforts undertaken by our colleagues. In the first part of the article, we presented four case studies covering quick-and-dirty and mock-up prototypes. In this final part, we add three more case studies of throw-away, exploratory and design-driven prototypes.

Throw-Away Prototyping

A throw-away prototype is any prototype that you create for the purpose of understanding – hence a quick-and-dirty prototype is by definition a throw-away, and a mockup prototype may either be a throw-away or be used in an evolutionary manner. The purpose of throw-away prototyping is to form the basis for discussion with a potential customer as to actual requirements. A throw-away typically is only partially functional. "Throw-away," then, is a descriptive dimension that you can apply to all other kinds of prototypes.

The trick behind successful throw-away prototyping is to plan from the beginning to discard the code, and then actually to do so. This often means assigning a different team to the next stage of the project. As mundane as this sounds, it is the key issue. Effectively, throw-away prototyping serves the purpose of learning and practicing before the big event. And it allows the project leader to explore and evaluate more pathways toward a successful solution. Discarding code is a mindset: to be willing to rethink the style and architecture of the code, rather than to patch. It is usually more cost-effective, not less, because patching is a temporary idea – a band-aid.

© 1990 ParcPlace Systems

Case Study 5: Application Framework

"Ideas must work through the brains and the arms of good and brave men, or they are no better than dreams."
—Emerson

Project Description. The project goal was to create an executable demonstration of a sophisticated information management system based on hyper-media ideas. The project was done partially under contract, with the first implementation phase specifically described as an analysis prototype. The resulting framework had to support a prescribed set of functionality, which was determined via an end-user requirements survey and an analysis of a prior product that no longer served the customer's needs. Testing was to be done by the contractor, who would attempt to build applications within the framework – applications not previously described in detail to the designers. The project was intended to be accomplished within eight to ten weeks.

Of all the case studies we have presented, this is the only one that was contractually designated as a prototype; both parties (users and developers) agreed that all the code would be discarded for the next stage. This agreement was important because the objective was to get a result quickly, overriding any notion that the underlying architecture needed long-term stability.

Team Composition. The team consisted of one project leader and three programmers.

Process Setup and Execution. Initially, the project consisted of one expert framer ("carpenter") to do all of the framework-level programming, and one documentor/project leader. The project leader's role was to review the design of each new class of objects, review all on-line documentation for readability and accuracy, update the functional specification each day based on current coding and on-line documentation, and control any deviation from plan.

After the first week of prototyping, one tester/programmer was assigned the job of building examples using the project objects to see if they were reusable outside the project. The examples were initially assigned by the project leader to test individual objects. This tester was in essence an apprentice who, in doing the testing, learned about the project goals, design ideas and object-oriented framework for development.

After the third week of prototyping, when the initial application framework was in place, a second tester/programmer was added to implement additional information structures that would fit into the framework (i.e., to test the framework).

On a daily basis, the documentation was revised until it could ultimately serve as the functional specification, consisting of:

- basic idea for application metaphor
- basic data structures
- basic information structures
- editing operations on data and information structures
- user-interaction operations

The project leader served as project tracker and documentor. Documentation was also done on-line, so that every class and every method was fully documented. A hardcopy manual writer was available to map this on-line documentation to a well-formatted reference guide.

Outcome. After ten weeks, the following deliverables for the throw-away prototyping project were completed:

continued on page 8 . . .

- an executable prototype that was useful as a scaled-down version for part of the fully envisioned product
- a code-reference document (generated from the on-line documentation)
- a draft functional specification that had been revised throughout the prototyping period

The next step of the project is to write a design document and create the design prototype. The analysis prototype code was abandoned as planned. The expectation is that the design prototype can be implemented, evaluated, and probably evolved into the product code.

Evolutionary/Exploratory Prototyping

Exploratory programming generally is associated with a style of software development in which the programmer starts with a general idea of the desired outcome, but with no particular specifications or complete requirements. The act of creating the software answers the question of what to build as well as how to build it.

The process of programming is to get started and incrementally make modifications – molding a figurine out of clay rather than chipping a statue out of marble. In the first case, you can afford to make a mistake; in the second, you cannot, because the specification would have to change. In the first case, you do not necessarily need to have advance knowledge of what you will have when you are done; in the second, you need a stable plan.

In exploratory prototyping, getting done fast is not the critical element as it is with quick-and-dirty prototyping. Mockup prototyping is usually more results-oriented in the sense that the illusion to be created on the computer is known (or at least well-understood) before the programming begins. The ability to entertain radical and sweeping changes is an important aspect of exploratory prototyping.

Exploratory prototyping is typically used in research laboratories, where each stage of developing the prototype serves as a springboard for the next evolution in the research project.

Case Study 6. An Expert System

"To be conscious that you are ignorant is a great step to knowledge."

—Disraeli, *Sybil*

Project Description. The goal was to design an operator's associate to aid in the control of dynamic systems such as nuclear power plants and satellite ground control systems. In essence, a computer program was to be designed to act as a second controller, supporting the human operator of such systems.

To do so, the computer-based associate needed a way to interpret the human operator's activities, and then to infer the operator's intentions in order to take over low-level activities during periods of heavy load. This was an academic research project whose primary focus was to formulate, design, implement and validate a theoretical model of an intelligent operator's associate.

Team Composition. The team consisted of one principal investigator and one student researcher/programmer.

Process Setup and Execution. There was no well-defined, *a priori* software specification for the desired system. Rather, system construction was an exploratory process, with theory, design and implementation closely intertwined. At each step of exploration, a variety of alternative system designs were evaluated and the evaluations were documented in writing for review by research peers.

In the course of developing the prototype, an architecture for a very large computer-based reasoning system was implemented (replicated from a description of another research result) during a three-month time period.

Because the software was designed using an object-oriented language, the programmer was able to map the theoretical understanding of the team members directly to the information represented in the computer. The resulting papers therefore represented documentation about the implementation design.

Outcome. The deliverables from this project were publishable papers. The software was itself intended only as a usable prototype that could be tried out by the experimenters. Because the team understood that they were building a prototype with which they could gain considerable insight into the design of the intelligent computer assistant, they were able to make good use of the evolving software. Nonetheless, the software had to be maintainable and understandable so that it could form the basis for further research by other student researchers. It was quite successful in this regard, forming the basis for two doctoral theses and one master's thesis.

Design-Driven Prototyping

A design-driven prototype is never a throw-away. The idea is to create a preproduction model of a system. The model is as complete and as perfect as it can be made to be, and is then sent out to be test-driven by end-users. This prototype is often referred to as an *operational prototype* as contrasted with a *development prototype*. The prototype is considered useful and usable by the actual target customer. It is placed in production with a small number of customers for first-stage evaluation, revised if necessary (iteratively), and then redistributed.

Case Studies 1 and 2 (published in the April issue) made it to this stage, despite the fact that they were not driven from a stable design and the code should have been discarded. Hence, they were remarkable failures.

Case Studies 4 and 6 were both about research projects, used only by members of the research laboratories; the restraint shown in not distributing the prototypes is a statement of their individual successes.

Case Study 7 is a good example of a design-driven prototype. The intent of the project was to create something that could be put in real users' hands. Partially because the software was distributed at no cost, and all source code to the system has also been made freely available, the user community has been quite satisfied with the prototype status of the software.

Case Study 7. Music Composition

"Never resist temptation;
prove all things;
hold fast that which is good."

—George Bernard Shaw
Maxims for Revolutionists

nents of the post office.

The pass at OOD for the post office would:

- detail the precise programming language interfaces for the collection of mailboxes, the postal clerk, and the "other post office objects;"
- create the programming language algorithm for the internal structure of the post office at the top level (i.e., how the post office will manipulate the collection of mailboxes, the postal clerk, and the "other post office objects").

Assuming that the postal clerk is a fairly simple object with life, OOP would create the entirety of the source code for the postal clerk.

Returning to the top level of abstraction (the EMS):

- The post office, message carrier, and "other objects" would be the "analysis objects" for the EMS.
- With respect to the EMS, the collection of mailboxes, the postal clerk, and the "other post office objects" are "design objects."
- The post office, message carrier, and "other objects" are the "interface objects" with respect to the EMS and the rest of the EMS system.
- Message objects are "commuting objects" with respect to the top-level EMS and the post office.

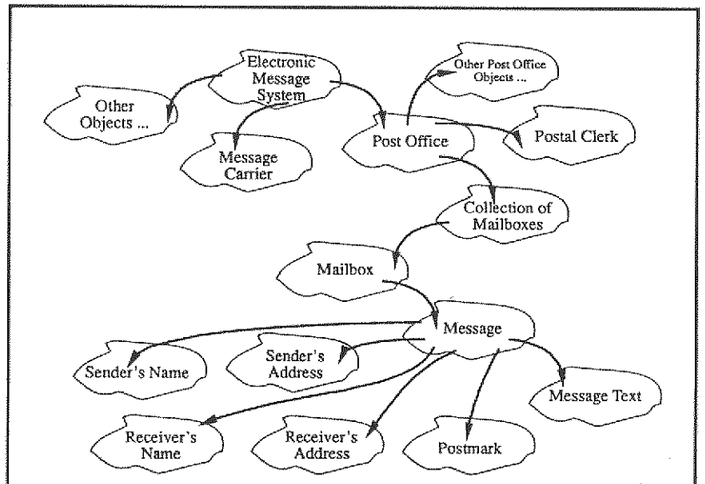


Figure 1: A graphical representation of a partial object-oriented decomposition of a simple electronic message system shows several levels of abstraction.

When to Use OORA, When Not to Use It, and When to Stop Using It

Often, OOD and OOP are all that is necessary for a product. However there are times when OORA is justified, such as:

- When the overall size of the project is very large (e.g., significantly larger than 2,000 lines of code).
- When the software product being developed is critically important.
- When the customer, or management, dictates that a conventional, waterfall life-cycle must be followed – even though this is inappropriate for an object-oriented approach.

Given a recursive/parallel approach, a large (or critical) project, and the fact that OORA was used at least at the start of the effort, a software engineer would want to know when to stop using OORA on the project. The following are situations where one can consider not using OORA:

- if the component (or subcomponent) is small (e.g., ≤2,000 lines of code)
- if the component is slightly larger than about 2,000 lines of code, and the component is non-critical
- if the component (or subcomponent) is recognized as a pre-existing library component
- if the component (or subcomponent) is a relatively simple variation on a pre-existing library component

Consistency in the Object-Oriented Life Cycle

Object-oriented thinking is more consistent than traditional structured thinking. This both simplifies and complicates matters.

Good News: Since they are more consistent in thinking, the "gap" between OORA and OOD is very narrow, when compared to the "gap" between structured analysis and structured design.

Bad News: This high level of conceptual integrity makes it harder to distinguish between OOD and OORA.

Differences Between OOD and OORA

Even though I mentioned some of the differences between OORA and OOD in another article, it might be useful to repeat them here:

- Although they share many graphical techniques in common, there are differences. For example, OOD uses program unit graphics, whereas OORA does not.
- Programming language issues are much more prevalent in OOD than in OORA. Client visibility is higher in OORA than in OOD.
- Not all objects identified during OORA will become code software, whereas many (if not all) of the objects identified during OOD will become code software.
- OORA does not identify all of the objects in a system. Additional objects will be introduced during the OOD process.
- The larger the overall project, the higher the ratio of design objects to analysis objects.

Conclusion

Since this is only a short article, and not a book, I have left a great deal out. However, what I have presented should give you a "flavor" of the recursive/parallel life-cycle.

Edward V. Berard is founder of Berard Software Engineering, Inc. He may be reached at (301)353-9652, 18620 Mateney Road, Germantown, MD 20874.

ous analysis and/or design is often mandatory (i.e., we will be required to "analyze a little, design a little, implement a little, and test a little" even if the product is "only a few hundred lines of code").

Accomplishing the Recursive/Parallel Approach

It is useful to establish some general guidelines for what we mean by each part of "analyze a little, design a little, implement a little, and test a little."

The "analyze a little" step requires that we:

- understand the requirements for the product (or component);
- propose a "high-level" solution for these requirements which involves identification of the major components (or subcomponents);
- demonstrate that the proposed solution meets the client's needs.

The "design a little" step requires that we:

- precisely define the interfaces to the components (or subcomponents);
- make decisions about how each component (or subcomponent) will be implemented in the selected programming language;
- identify any necessary additional program units;
- describe any necessary programming language relationships (e.g., nesting and dependency).

The "implement a little" step requires that we:

- implement the programming language interfaces for each of the components (or subcomponents) *[note that this substep, and the next one, allow one to use a programming language in the form of a design language];*
- implement the algorithm that describes the interactions among the components (or subcomponents);
- implement the internals of components that will not be further decomposed.

The "test a little" step requires that we:

- compile any code produced (or selected) as a result of the "implement a little" step; this should check things like syntax, some semantics, and some interfaces;
- carry out any dynamic testing (machine-executable testing) that is possible;
- perform any necessary (or required) static testing.

Of course, regardless of where we are in the recursive/parallel life-cycle, we must take care to ensure the quality of the product. Software quality assurance (SQA) is an on-going effort.

Analysis Objects, Design Objects, Interface Objects, and Commuting Objects

Object-oriented requirements analysis (OORA) stops at the user interface to a [potential collection of] system[s] of objects. It is the job of object-oriented design (OOD) to precisely define the interface for each system of objects, and, if the system of objects is small enough, to define the internal structure (architecture) of each system of objects.

It is very likely that some objects identified during analysis will not become code software. We refer to these objects as *analysis objects*.

It is also very probable that we will identify objects during design that were not addressed in analysis. We refer to these objects as *design objects*. Most of these objects will indeed become code software.

One or more objects may lie on the user interface to the system of objects (i.e., taken in combination, they *are* the user interface for the system of objects). We refer to these objects as *interface objects*.

There will be objects that were mentioned in analysis, *and* are needed for the proper behavior of the system of objects. In effect, these objects are inputs to the system of objects, outputs from the system of objects, or both. We refer to these as *commuting objects*.

A Simple Exercise

Let us now consider a simple example. We have been tasked with the creation of a simple "electronic message system" (EMS). What are some example scenarios for the object-oriented development of such a system?

One scenario (see Figure 1) might be to view the EMS as composed of:

- a Post Office
- a message carrier
- and other EMS objects ...

The Post Office is in turn composed of:

- a postal clerk
- a collection of mailboxes
- and other post office objects ...

One of the components of a collection of mailboxes is an individual mailbox.

One of the components of an individual mailbox would be an individual message.

The components of a message could be:

- the sender's name
- the sender's address
- the receiver's name
- the receiver's address
- the postmark (i.e., the date and time the message was sent)
- and the actual text of the message.

Given the previously (partially) described EMS, the first pass at OORA would:

- identify the post office, message carrier, and "other objects" as being components of the EMS;
- describe the environment in which the EMS will be used; and
- describe the conceptual interfaces of the EMS, post office, message carrier, and "other objects".

The first pass at OOD would:

- detail the precise programming language interfaces for the EMS, post office, message carrier, and "other objects";
- create the programming language algorithm for the internal structure of the EMS at the top level (i.e., how the EMS will manipulate the post office, message carrier, and "other objects").

Assuming that the message carrier is fairly simple, the first pass at OOP would create the entirety of the source code for the message carrier (and any other very simple objects at that level).

Assuming a certain level of complexity, one of the second passes at OORA would identify the collection of mailboxes, postal clerk, and "other post office objects" as being compo-

Project Description. The goal was to create a music-composition system consisting of a music-description language, a music-representation framework, a scheduling- and realtime-performance framework, a set of user-interface tools for building music applications, and a set of built-in applications.

An existing music-composition system had already been built using a non-object-oriented approach. This prior system provided the specification for the desired functionality. The design was therefore basically available, but the specifics had to be evolved based on the object-oriented technology now available.

Team Composition. A project leader took on the task of defining the music-representation framework and tools for handling the music-description language. He was located remotely from the other four participants. The other team members contributed parts of the scheduling and applications code, thereby testing the framework and tools.

Process Setup and Execution. Once the project leader defined the framework – and froze the design, implementation and documentation – the others contributed schedulers and user interfaces according to the protocols of the framework. The project leader served as the systems integrator to combine all of the contributed components. No group design reviews were conducted.

The result was a prototype in the sense that it is not a commercial product, nor was it intended to be. Rather, it was intended to serve as an evolving system in which a group of music composers could investigate their ideas for computer support of the composition process. Other than to a small and familiar community of computer-based music composers, the system was not intended for general distribution.

Outcome. The current user base contains 50 serious uses out of 100 distributed copies. There is a users' group that shares ideas about the system, and shares the musical results of its use. Because the group understands that the system was not intended to meet the quality of a commercial product (nor to command commercial prices), the prototype status continues to be acceptable. A revision to improve source-code documentation is currently in process.

Summary of Reasons for Success or Failure of Prototyping

We have given seven examples of prototyping projects, some of which succeeded and some of which failed. What distinguishes the successes from the failures?

On successful projects, the prototypers:

- *understood the purpose of the prototype and used it only for that purpose*
- *took the time to understand the technology to be used and the prototyping process*
- *assembled an appropriate team to do the prototype: project leader / documenter, framework creator, and framework users / testers (i.e., the application programmers)*
- *did reviews / evaluations of team and deliverables*
- *involved target users early in the process*
- *were willing to iterate over the prototyping process in order to better understand the underlying architecture*
- *set proper evaluation criteria at the beginning of each stage of prototyping, and firmly based completion on that criteria*
- *took the time to look for reuse opportunities, by using existing code and by adding to a library of reusable code*
- *built prototypes based on an existing approved company library of reusable code, controlled by an assigned librarian*

Where the project failed, the prototypers:

- *did not understand what a prototype is and how it should be used*
- *did not structure the team with the appropriate personnel for prototyping, reuse support, analysis and design documentation (i.e., did not understand the process well enough to set up the team correctly)*
- *did not separate the development and evaluation of the analysis prototype from that of the design prototype*
- *did not know when to stop evolving a prototype and start anew (i.e., they over-extended the process)*
- *did not know when to continue in order to get a result that met the desired evaluation criteria (i.e., they prematurely terminated the process).*
- *did not identify existing or new reusable assets*
- *did not use supportive object-oriented development environments or tools, but only an object-oriented language*
- *believed that a reasonable prototype was an acceptable product*

Recommendation

Our recommendation is to use prototyping in order to obtain true end-user requirements (via an analysis prototype), and to achieve architectural clarity (via a design prototype). At least five team members are needed: project leader/documentor/closer, programmer as framer or carpenter, framework tester, components/reuse tester, and the end-user. Additional aspects of team structure, including performance metrics, reward structure and career paths, will be discussed in a separate article.

Dr. Adele Goldberg is President and Chief Executive Officer of ParcPlace Systems. Adele spent 14 years as a researcher and manager at Xerox Palo Alto Research Center, prior to founding ParcPlace. She has served as president of the ACM Systems Software Award.

Kenny Rubin is a computer scientist at ParcPlace systems, where he has been a primary curriculum developer of three of the ParcPlace Systems Smalltalk-80 training courses. In addition, he has served as lead instructor for many of the ParcPlace Systems courses taught over the past two years and has written on object-oriented technology, artificial intelligence and human-computer interaction.

Both authors may be reached at ParcPlace Systems, Inc., 1550 Plymouth Street, Mountain View, CA 94043; (415)691-6700.

Analysis & Design/ Edward V. Berard

Understanding the Recursive/Parallel Life-Cycle

In my previous article, I mentioned that a recursive/parallel approach is the one which is most appropriate for the object-oriented life-cycle. What I will attempt to do in this article is to provide you with a better understanding of the recursive/parallel life-cycle.

In researching the recursive/parallel approach, I discovered that even though this approach to the object-oriented life-cycle has been in use, and evolving, for at least 8 years, no definitive article has been written on the topic. To be sure, various aspects of the recursive/parallel approach have been discussed in articles, seminars, and net messages. Further, it has actually been used on quite a number of actual development efforts.

Unfortunately, simple descriptions such as "analyze a little, design a little, implement a little, and test a little," are often more confusing than they are enlightening. In addition, there is a strong tendency to confuse the recursive/parallel life-cycle with rapid prototyping approaches. This article is an attempt to reduce, if not eliminate, this confusion.

Historical Background

"Chaos is a friend of mine."
— Bob Dylan (quoted in *Newsweek*, December 9, 1985)

Years ago, the state-of-the-practice was to "write a bunch of code, and then test and debug it into a product." Once all the code was written, and "debugged," it was often necessary to document [the so-called "design" of] the product. In short, the idea was to "code first, and think later."

In an attempt to improve the quality of both the product and the process, the concept of a software life-cycle with explicit analysis and design phases emerged. The most prevalent life-cycle approach was the *iterative waterfall*, or *cascade* model. This approach required that "all analysis" be completed and verified before design could begin, and that "all design" be completed and verified before coding could begin. While the waterfall life-cycle was generally an improvement over the previous chaotic attempts to develop software, it was not the final word.

When Grady Booch adopted the work of Russell J. Abbott to Ada software development (G. Booch, "Describing Software Design in Ada," *SIGPLAN Notices*, Vol. 16, No. 9, September 1981), he referred to the process as "object-oriented design," and demonstrated it using several simple examples.

Later, when people attempted to apply Booch's method to medium-sized projects (e.g., up to around 30,000 lines of code), they found that the actual development effort required repeated applications of the approach. Specifically, in decomposing the problem, they found that Booch's OOD could be re-applied to some (or all) of the components uncovered via (object-oriented) decomposition.

Booch, and others, began referring to the OOD approach as "design a little, code a little." This described a process wherein some object-oriented decomposition (design) was accomplished, followed by some coding. Then, depending on the complexity of a particular component, it may have been possible to re-apply the OOD process to that component. At my former company (EVB Software Engineering, Inc.), we expanded this approach to include necessary testing (i.e., "design a little, code a little, test a little.")

Later, because the word "code" seemed to cause concern among some managers (and customers), "code a little" was replaced by "implement a little." In 1986, when we, and others began focusing on object-oriented requirements analysis (OORA), the phrase "analyze a little" was added.

Within the Ada community there has always been a significant effort to use Ada (a coding language) as a design language (DL). This effort encouraged the recursive/parallel approach since the "implement a little" allowed for the use of the programming language as part of the "design."

While all of the above makes a nice story, it obscures the rigor within the recursive/parallel life-cycle. We must now take a closer look.

Decomposition

Many life-cycle approaches employ some form of decomposition. The concept is simple enough:

- select a piece of the problem (initially, the whole problem)
- determine its components (using the mechanism of choice)
- show how the components interact
- repeat the previous steps on each of the components until some completion criteria is met

The most familiar example of this approach is classic functional decomposition. Decomposition techniques are *top-down* approaches. A top-down approach has the following characteristics:

- It progresses from the general to the specific, and, by induction, from *what* to *how*.
- At any point in the process we always have a complete (although possibly limited in detail) description of the system. (We often refer to the deliverable from a major step in a top-down approach as a *layer*, where a layer is a complete description of the system at a given uniform level of abstraction.)
- Items at higher levels of abstraction treat items at lower levels of abstraction as "black boxes".
- While items at higher levels of abstraction may be aware of items at lower levels of abstraction, items at lower levels of abstraction are unaware of items at higher levels of abstraction.

Many people often confuse *top-down* with *functional decomposition* or *structured* approaches. While these approaches are indeed examples of top-down thinking, they are implementations of the concept, not definitions.

It is possible to decompose a system in an object-oriented manner:

- view the problem as an object or system of objects;
- identify its major parts in terms of interacting objects;
- show how the component parts (objects) interact to provide the characteristics of the composite object (or system of objects);
- repeat the previous steps on each of the component objects until some completion criteria is met.

We note that object-oriented decomposition differs from functional decomposition primarily in the way information is localized (i.e., around objects *vs.* around functions). Other examples of noticeable differences include:

- Functional decomposition components have *verb* names, while object-oriented components have *noun* names.
- Objects are typically more complete abstractions than are functions.

Composition

Most conventional object-oriented programming (OOP) approaches are compositional approaches, in that they:

- survey the problem, attempting to identify necessary components;
- select components from an existing "library of components" and/or create new components as necessary;
- assemble the components to form larger components.

If the larger component satisfies the complete problem, then stop; otherwise, continue to assemble and merge components until the problem is (or appears to be) solved. Obviously, in OOP, the components will be objects.

Composition techniques are *bottom-up* approaches. A bottom-up approach has the following characteristics:

- It progresses from the specific to the general and, by induction, from *how* to *what*.
- At any point in the process we usually do not have a complete description of the entire system (with the possible exception of the description of the original problem).
- Most often, we are building black boxes – not decomposing them.

A bottom-up process need not be chaotic, and is most appropriate for rapid prototyping or research efforts. Further, a composition process may be entirely adequate if the problem is small, well understood, and non-critical.

Reusability and Compositional Techniques

Reusability implies that we must be using composition techniques at some point in the life-cycle. We may arrive at the composition process in one of two ways:

- We initially select reusable components and combine them to solve a problem.
- We decompose a problem to a point where we can easily and accurately identify existing reusable components, and then select these components.

And Now, the Recursive/Parallel Process

In general, the recursive/parallel approach (a top-down approach) stipulates that we:

- decompose a problem into highly independent components;
- re-apply the process to each of these components to decompose them further (if necessary) – this is the *recursive* part;
- accomplish this re-application of the process simultaneously on each of the components – this is the *parallel* part;
- continue this process until some completion criteria is met.

The process that we will be applying, in whole, or in part, to each of the components is "analysis, followed by design, followed by implementation, followed by testing."

For the recursive/parallel approach to be effective, the components must be as highly independent of each other as possible. Well-designed objects tend to be much more independent of each other (in an overall system) than are well-designed functions.

At this point, we need to make several clarifications:

- Although a recursive/parallel approach may be used with functions, it is recommended, for best results, that it be used with objects.
- We will emphasize reusability in our recursive/parallel approach.
- As opposed to a "pure" top-down approach, we will also use composition techniques with our recursive/parallel life-cycle.
- The larger the product (or component) the more top-down (and decompositional) will be our approach.
- The smaller the product (or component) the more likely it will be that we use compositional techniques

Occasionally, when people first see "analyze a little," they think that this implies a "sloppy" (or an inappropriate amount of) analysis. This is not true. As with any other life-cycle approach, some decision must be made at the beginning of the project as to which details must be considered first, and which can be considered later.

The recursive/parallel life-cycle is no different. "Analyze a little" means that one must identify the details which are appropriate to the current level of abstraction. Further, the analyst must make sure that details left for later can truly be ignored until later.

We do not necessarily have to perform an entire "analyze a little, design a little, implement a little, test a little" process with each new component. For example, if the size of the component is *small*, we may not need to perform analysis or design. We may recognize a component as being one that is already in our reuse library, or is a relatively simple modification of a pre-existing component. In this case, we may simply reuse (or slightly modify) the pre-existing component.

Size and Critical Nature of a Software Product

If a software product (or component) is small enough (e.g., ≤ 2,000 lines of code), then all that may be needed is "implement a little, test a little." However, size alone is not a determining factor.

Even if a software product (or component) is relatively small, it may be a highly critical application (for instance, the kernel of an operating system or an embedded flight control application in a commercial aircraft).

For critical applications, some seri-

continued on page 12 . . .

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ **Yes,** I want to be plugged into the latest thinking and developments in object-oriented technology. Enter me immediately as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	2 years (24 issues)
☐ ~~299~~ $249	☐ ~~$598~~ $498

outside the US — add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT — US currency drawn on US bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card # _____ Exp. date _____
Signature _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____

Send Subscription Orders To: HOOT, P.O. Box 3000, Dept. HOT, Denville NJ 07834
For immediate service—call the Hotline 212-274-0640 or Fax 212-274-0646

HOTLINE on
Object-Oriented Technology
The International Newsletter

588 Broadway, Suite 604
New York, NY 10012
(212) 274-0640

HOTLINE on
Object-Oriented Technology
The International Newsletter

June 1990 **Volume 1, Number 8**

A Taste of the USENIX C++ Conference

Mary H. Wells

Although the local nickname for the new Marriott Hotel in San Francisco is "the Jukebox," the second annual Usenix C++ Conference held there on April 9–11 was not a song and dance. Approximately 550 attendees came to learn, to discuss, to share experiences, and to offer suggestions for improvements to the language.

Starting with the keynote address, given by **Adele Goldberg** from ParcPlace Systems, a common theme appeared. We believe object-oriented technology can help us, we have tried C++ and found it useful, but we have also discovered some problems.

Not the least of the problems was the length of time to really understand object-oriented concepts and use them effectively in C++. According to Goldberg, learning O-O concepts was hard because of

- a lack of standard terminology and nomenclature
- a lack of shared expectations about object-oriented design
- the requirement of a hybrid language, such as C++, to sometimes break the object-oriented metaphor.

She further stated that building on a knowledge of the C language to learn C++ was part of the problem, rather than part of the solution. It was easier to fall back on old habits. She advocated first learning the paradigm, then learning the language.

Addressing another area of concern, **Bjarne Stroustrup**, developer of C++ at AT&T Bell Laboratories, discussed the latest version of "Exception Handling for C++," by Stroustrup and **Andrew Koenig**. There were no new ideas; instead, the improvements made were better approximations for the same ideas presented in earlier papers.

An exciting development was the appearance of more technical papers concerning large scientific and engineering applications. One interesting project at Los Alamos National Laboratory was described in the paper "Experiences in Writing a Distributed Particle Simulation Code in C++."

The goal of the project was to take a 20,000+-line simulation application written in Fortran and write it in an object-oriented manner to:

- simplify and accelerate its use on large parallel computers
- match the numerical model more closely to the physical world
- evaluate the usefulness of C++ for scientific computing
- observe some ways in which the language might be improved.

The authors of the paper noted that C++ delivered on its promise to build modular and easily-extended physics codes. However, several areas of improvement suggested themselves. In particular, function calls in C++ degraded performance by a factor of 1.7 on a Sun 4. The difference was even greater after optimization (a factor of 2.2). They called for better C++ compiler technology to increase optimization.

As the title suggests, this is just a taste of the conference. More information is contained in the conference proceedings. Considering the wealth and diversity of the papers presented, it is obvious that object-oriented technology in general, and C++ in particular, is no longer limited to use by the innovators and early adopters, but is rapidly passing into the mainstream of the computer industry. For those of us who have been around since the early days, this is the most exciting development — and a reason for us to sing and dance. ■

Mary Wells is Technical Director of Semaphore Training, with six years of experience in object-oriented technology. She may be reached at (508) 794-3366.

INSIDE...

From the Editor 2

Databases 3
Comparing Relational & O-O DBMS

Project Management 5
Case Studies in Prototyping, Pt. 2

Analysis & Design 9
Object-Oriented Requirements Analysis

Company Showcase 12

Publicity Corner 13

Product News 16

Calendar 19

FROM THE EDITOR/JEFF MCKENNA

This issue of the HOTLINE looks a bit like a soap opera. Each of the major articles builds on information presented in earlier issues, as our authors delve deeper into their topics.

Adele Goldberg and **Kenny Rubin** discuss the organization of software development teams that utilize object-oriented technology. They propose an organization that explicitly recognizes that software reuse is a major goal and contains organizational units that are directly responsible for promoting and encouraging reuse.

Edward Berard builds on his earlier articles by discussing object-oriented requirements analysis. He identifies this process as a reasonable way to define a project using objects, and he also points out ways to identify and integrate reusable components early in the development process.

Robert Martin continues his discussion of object-oriented databases by discussing how composition, flexibility and extensibility are factors in O-O database systems that differentiate them from classical database systems.

Mary Wells of Semaphore Training gives us a brief glimpse of the Usenix C++ Conference recently held in San Francisco.

The Publicity Corner continues to expand as we see more and more exposure of "object-orientedness" in the press.

I just got off the plane from Boston, where I attended the first two days of SCOOP-East. Some 450 attendees gathered at the Wang Institute to learn the latest about objects from an impressive group of industry leaders. Less than half of the attendees were currently using the technology, but they were strongly considering it.

What I noticed from the attendees was a sharper focus on the use of the technology. While many did not claim to understand exactly what was going on, there seemed to be a general acceptance of objects as the way to go. The attendees were less dogmatic about positions, i.e., which language to use, and appeared to recognize that the technology is still immature. Folks asked about using objects on mainframes and wanted to understand the difficulties of analysis and design at a very practical level.

The management folks I met appeared to recognize that there's a problem in software development. They were saying, "Okay, we have a problem. Now let's get on with it."

I see these developments as healthy. People are rolling up their sleeves and have recognized that objects are not a 'silver bullet' but that they do have considerable promise. By sharing with each other as we take those first difficult steps, we can all profit.

Jeff McKenna
Editor

Advisory Board
Jim Anderson
Digitalk, Inc.
Tom Atwood
Object Design, Inc.
Grady Booch
Rational
Larry Constantine
Consultant
Cort DeVoe
IBM
Adele Goldberg
ParcPlace Systems
Bertrand Meyer
Interactive Software Engineering
Reed Phillips
Knowledge Systems Corp.
Trygve Reenskaug
Taskon A/S
Bjarne Stroustrup
AT&T Labs
Dave Thomas
Object Technology International
Zack Urlocker
The Whitewater Group
Eugene Wang
Borland International

Richard P. Friedman
Group Publisher
Margherita Monck
General Manager
Elisa Varian
Production Manager
Susan Culligan
Typography & Design
Melanie Younossi
Marketing Manager
Gregory Tsichlis
Circulation Manager

The Hotline on Object-Oriented Technology (ISSN #1044-4319) is published monthly by SIGS Publications Inc., 588 Broadway, New York, NY 10012 (212) 274-0640. ©Copyright 1989 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher. Mailed First Class. Subscription rate—1 year (12 issues) $299. Foreign and Canada $323. Single copy $30.
POSTMASTER: Send address changes & subscription orders to HOOT, P.O. Box 3000, Dept HOT, Denville NJ 07834
Submit editorial correspondence to Marie Lenzi, 210 East 86th Street, New York, NY 10028.

INSTANTIATIONS ANNOUNCES APPLICATION DEVELOPMENT TOOLS FOR SMALLTALK

Mike Taylor
Instantiations, Inc.
503-242-0725

PORTLAND, OR - MAY 21, 1990: **Instantiations Inc.** announced today a powerful new set of tools to aid developers of Smalltalk-80™ applications. The product, called **Application Organizer 80™**, gives programmers a major advantage in the design, management, and delivery of their Smalltalk-80 applications.

Smalltalk is widely recognized for its tremendously productive development environment and support for rapid prototyping. Application Organizer 80 extends Smalltalk's capabilities by increasing support for the engineering and delivery of end-user applications.

Adding powerful new capabilities and tools, the new product assists programmers in optimizing designs, modularizing code, and minimizing the size of finished applications. "These tools help us design and deliver better Smalltalk applications for our clients. Our designs are cleaner and final code size is significantly smaller," said Smalltalk pioneer Allen Wirs-Brock, Instantiations' Vice President of Engineering.

Application Organizer 80 includes the tools listed below:

- The Image Generator
- The Package Organizer
- The Inheritance Browser
- Workspace Enhancements
- The Package Browser

Application Organizer 80 is priced at $195 for a single-user license and may be ordered directly from Instantiations. Apple Macintosh and 386-based MS-DOS computers are currently supported. Other platforms are under development. Shipments will begin in June 1990.

Instantiations is an engineering services firm specializing in object-oriented experience. The company is located in Portland, OR, and provides training, consulting, and development services to companies around the world.

Calendar

The HOOT calendar presents conferences and meetings that focus exclusively on object-oriented technology. It should be noted that many other other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to Marie Lenzi, 210 East 86th Street, New York, NY 10028.

June 6–8, 1990: Eurographics Workshop on Object-Oriented Graphics – Konigswinter, Federal Republic of Germany.

Workshops cover standardization and object-oriented languages for graphics.

Contact: Workshop, CWI, Kruislaan 413, 1098 SJ. Amsterdam, The Netherlands. Tel. 31-20-592-4058.

July 9–11, 1990: Eighth British National Conference on Databases – University of York, UK.

Emphasis on O-O databases. Keynote, William Kent, HP (USA). Technical sessions, lectures and exhibits.

Contact: Dr. Alan Brown, BNCOD-8, Dept. of Computer Science, University of York, Heslington, York Y01 5DD, UK, 011-44-0904-432746.

Sept. 14–15, 1990: Symposium on OOP Emphasizing Practical Applications – Marist College, Poughkeepsie, NY.

Keynote: Bertrand Meyer. Tutorial: Richard Wiener. Technical sessions.

Contact: James Ten Eyck, Marist College, Poughkeepsie, NY 12601, (914) 471-3240 Ext. 606.

June 26 – 29, 1990: TOOLS '90 – Second International Conference and Exhibition. CNIT-Paris, France.

Program Chair: Jean Bezivin. Conference Chair: Bertrand Meyer. Tutorials, workshops, technical sessions.

Contact: TOOLS '90, SOL, 14, rue Jean Rey 75015 Paris, France. Tel. 33-1-40-56-0358.

July 16–18, 1990: SCOOP-Europe – Imperial College, London, UK. Co-sponsored by *JOOP* and BUCEC.

Keynote-Adele Goldberg. Tutorials, panels, products and exhibits.

Contact: Chantal Olsen, BUCEC, 72 Tyng Rd., Tyngsboro, MA 01879, (508) 649-9731.

Sept. 24– 27, 1990: C++ At Work '90 – Wang Institute of Boston University. Sponsored by THE C++ REPORT & BUCEC.

The International C++ Users Group Conference. Technical sessions, tutorials, panels, BOF & exhibits.

Contact: Bob Daniels, (508)649-9731.

ParcPlace Announces Availability of Objectworks for Smalltalk-80 for Hewlett-Packard Apollo Series Workstations

Helene Tannor
ParcPlace Systems
(415) 691-6728
or
Ellie Victor
Cunningham Communications
(408) 982-0400

Mountain View, California, May 15, 1990 - **ParcPlace Systems** today announced the availability of **Objectworks® for Smalltalk-80™** for HP's Apollo Series workstations, which allows Apollo Series 2500, 3500, and 4500 professional programmers to tailor applications quickly and easily under the X-Window System™ .

With the addition of the Apollo workstations, *Objectworks for Smalltalk-80* Version 2.5 is available for more hardware platforms than any other O-O system currently on the market, including the 80386-based PS/2s® and compatibles; Macintosh® computers; Sun workstations; DECstations™ ; and the HP 9000™ Series 300. The product also runs under most popular operating systems, including UNIX®, MS-DOS®, and Macintosh.

Objectworks for Smalltalk-80 provides a new level of portability to all of the supported hardware platforms and operating systems. Programmers can build applications and move application binaries from one system to another without any code recompilation, saving time and costs.

"We are excited that ParcPlace Systems is bringing *Objectworks for Smalltalk-80* to our family of desktop workstations," said Mike Gallup, marketing development manager, HP Workstation Group. "It is sophisticated enough to meet the demands of our customers, who develop applications for electronic design automation, computer-aided software engineering, computer-aided design and automation."

A number of ParcPlace's customers have already decided to make their applications available on Apollo workstations. Ascent Logic Corp. plans to provide its System Engineering Tool on the Apollo Series 35XX to 4XXX workstations, in addition to the Macintosh II, Sun workstations, and DECstations. This product allows engineers to track and analyze decisions in systems engineering. "To our knowledge, we have one of the largest commercial applications running Smalltalk today," said John Cox, VP Sales at Ascent Logic. "ParcPlace's *Objectworks for Smalltalk-80* enables us to bring applications to market rapidly. We are pleased that ParcPlace is bringing this product to the Apollo—the more choices we can give our customers, the better."

Objectworks for Smalltalk-80 is penetrating business application development environments because corporations today are demanding a high level of portability across heterogeneous computing systems. The resulting portability of applications means greater flexibility for programmers and time and cost savings for companies.

Taking advantage of almost 20 years of research and technology, *Objectworks for Smalltalk-80* provides corporate software developers with integrated tools and reusable software components to create and deliver applications. It offers a complete software development environment, a uniform object-oriented development language with a dynamic compilation, and an extensive class library of more than 300 reusable software modules.

Price: $595 for MS-DOS and Macintosh; $3,495 for UNIX. Volume runtime agreements available; full technical support including customer hotline, upgrades to new releases, electronic BBS, and newsletters. In-house and on-site training available, all levels of object-oriented methodology and programming.

O-O Process Control Language

FORTH, Inc.
111 N. Sepulveda Blvd.
Manhattan Beach, CA 90266
(800) 55-FORTH

A fast real-time, event-driven, high-level control language for complex process control applications was introduced at Control Expo '90 by Elizabeth D. Rather, President of **FORTH, Inc.** Called the **Process Event Language,** this new English-like language will feature an object-oriented structure that closely follows the style of process and instrumentation diagrams.

"There is a one-to-one correspondence between the code and the objects being controlled," said Rather. "Instructions in the Process Event Language will use the same language that one process engineer would use to talk to another."

Founded in 1973, FORTH provides consulting services and a wide range of products to companies with demanding real-time applications.

Databases/Robert Martin

Comparing Relational and Object Database Management Systems, Part 2 – *Composites, Flexibility, and Extensibility*

In last month's article, we examined the advantages of object database's single paradigm programming, the fundamental difference between object and relational databases. This month, we explore other key differences, including support for composite objects, flexible transactions, and concurrency control, multiple storage managers, transparent cache management, and extensibility. Like single paradigm programming, these factors further enhance the improved performance and productivity that object databases provide.

Composite Objects

A composite object is a collection of other objects. The real world is full of composite objects, such as documents and purchase orders. A document, for example, is composed of various sections, paragraphs, footnotes, figures, and fonts. A purchase order consists of shipping and billing addresses, a set of line items, and prices.

Composite objects can be nested, or built with other composite objects. For instance, technical proceedings of a conference make up a composition of documents, while an account on a purchase order system would include all purchase orders for that customer.

When necessary, composite objects let you focus on the collection as a whole, allowing you to work at a level of detail that is appropriate for the task at hand. If you are creating a purchase order or document, you can focus on the individual line items and paragraphs. But when you print or mail them, you want to deal with the composite object. Compositions are a very intuitive and natural way to view and manage things.

Some object databases implement composite objects. Composites can be locked, stored, retrieved, and moved as a whole. This has three benefits—performance, productivity, and ease of database administration.

In the relational model, these is no way to identify composite objects. The closest mechanism is the "view," which can join together several relations to form a single table. But views have two major limitations that preclude their use with objects. Generally speaking, a multitable view cannot be used to update data, limiting it to read-only access. Also, some or all of this data in the view has been duplicated. At best, it is confusing.

Because there is no composite object support in relational systems, applications programmers have to assemble composite objects. Typically, this requires writing many slow and complex queries.

Advantages of Composite Object Support

Locking/Concurrency Control

Rather than individually locking each of its components, you can quickly and easily lock a composite object, which is the intuitive approach to managing items.

Activation/Deactivation

By transferring data in single large blocks rather than through many small exchanges, you can quickly activate, or load into memory, an entire composite with a single query. It is very simple to program a composite. Activation requires but a single line of code in an object database versus many pages of SQL in a relational system.

Ease of Administration

Composites form a natural basis for network administration. You put my POs on my machine and your documents on your machine. In relational systems, data is distributed by partitioning individual tables. As there is no notion of a composite object in the database, the administrator must figure out how to partition each table. These paragraphs go on this machine, those on the other machine. This is simply working at the wrong level of detail.

Clustering

Object databases support physical disk clustering. Objects frequently accessed together, such as a composite, are stored together on the disk so they can be retrieved in a single disk access. Using composites to cluster objects simplifies database administration and improves performance.

Flexible Transactions and Concurrency Control

All databases have a transaction and concurrency control model. A *transaction* is a single unit of work performed by an application. Either the transaction succeeds and all of the changes are committed to the database, or the transaction is aborted and none of the changes are made.

Concurrency is the ability for multiple users or applications to

share the same data in a controlled, predictable fashion. In particular, concurrency schemes determine what happens when two transactions try to modify the same data simultaneously — a conflict results.

There is a trade-off between concurrency and performance. No single scheme is best for all situations. The key is flexibility. While relational databases typically provide a single concurrency model, object databases provide a range of concurrency options, allowing you to choose the best one for your particular application:

Pessimistic. This is the "classic" database model. Each object needed within a transaction is locked, or prereserved, before it can be accessed. If another transaction has locked the data, the next transaction waits. If two transactions get into a "deadlock," in which each one is waiting for an object that the other has already locked, one of the transactions is aborted and the other allowed to continue.

Optimistic. In this scheme, it is assumed that transactions will rarely conflict. Objects are not locked as they are read. Instead, when the application commits the transactions, the database is checked to see if any other transaction has modified the objects. Conflicts are resolved by reordering updates, or by aborting one of the transactions. When conflicts are rare, this is much faster than the pessimistic model.

Long and short transactions. Some transactions are short and simple, such as debiting an account balance in an automatic teller machine. Other transactions, such as editing a document, can last hours or even weeks.

Nested transactions. Long transactions can be divided into a set of smaller, nested transactions. This is useful for two reasons. First, it makes it easy to implement an "undo" function, allowing users to back out of part of the changes they have made. Second, it checks for lock conflicts on a shorter time frame, limiting the impact of an aborted transaction.

Shared transactions. Today's window-based applications often have multiple processes running on behalf of the user. An editor might be running in one window and a spelling checker in another. Both processes need to join, or share, the same transaction.

Versioning. When two applications conflict, give them each their own copy, or version of the data on which to work. Later, they can determine how to merge the results.

Multiple Storage Managers

Object and relational systems take very different approaches to managing memory and storage. In many cases, this reflects the bias in relational systems to handling small amount of fixed-length record data, while object databases have been designed for larger, more complex data structures.

Storage managers provide the actual disk management. All modern storage managers support a distributed, two-phase commit protocol required to update multiple files simultaneously. Relational systems have a single storage manager, optimized for fixed-length records. Some object databases have multiple storage managers, each tailored for a specific task because different disk storage strategies are required for different types of data. Objects give you the flexibility to accommodate this.

Transparent Cache Management

Object systems employ sophisticated object caching schemes in which whole composites of objects can be "faulted," or activated into memory whenever the application touches any of the objects in the composite. This provides a very transparent programming mode, in which the programmers don't have to worry about which objects are in the memory. If an object is required, the database will get it. Transparent object caching simplifies application design and provides extremely high performance.

In relational systems, programmers have to explicitly request each record they need. This typically takes at least three queries, one to lock the data, one to read it, and one to write it. The programmers have to manage the in-memory copies themselves. This is both cumbersome and slower than the object approach.

Extensibility

Some object database systems have an open, extensible architecture that lets you provide your own implementations for various parts of the database system. Extensibility is the ultimate form of flexibility, and gives you the unprecedented power to tune the database for even the most performance-sensitive applications.

This extensibility is provided by packaging the database functionality as a set of C++ classes. You can create subclasses of these system classes to provide more specialized or higher-performance behavior.

Traditional relational databases are closed systems to which the programmer is not permitted to make any changes.

Next Month — The Mathematical Basis

Relational databases have a firm, mathematical basis. This has been an important factor in the acceptance of the relational mode. In the next column, we will examine the mathematical basis behind object databases and the impact that the mathematics have on the applications programmer. ■

Robert Martin, a founder and Vice President of Products for Ontologic, Inc., has led the commercial introduction of object database technology. Ontologic is the leading supplier of second-generation object database systems. For more information, contact Ontologic in Burlington, MA, at (617) 272-7110.

RASCAL SYSTEMS ANNOUNCES RASCAL, THE "CRAFTY" OBJECT-ORIENTED ALTERNATIVE

Rascal Systems, Inc.
4487 F. Park Drive
Norcross GA 30093
(404)717-0707
FAX: 717-0802

Rascal Systems announces release of **RASCAL** — "the crafty object-oriented alternative." RASCAL combines the techniques of OOP with the realization that systems development has to be flexible, fast, and convenient in order to be profitable. The net result is a total resource system that features the following:

- Predefined applications that developers can use as the basis for their own applications
- Large set of predefined object classes that substantially reduce amount of coding
- Code files portable across operating systems and hardware platforms
- Accelerated development of all applications
- Applications can be continuously upgraded and enhanced while still in use. Objects extended and changed without affecting other parts of system
- Objects may be debugged independently without being fully integrated into system
- Formal procedure for systems analysis and implementation
- Uniquely structured approach to implementation encouraging team programming
- Total environment without need for compilers and third-party libraries
- Predefines high-level objects built from other RASCAL object classes to further reduce programming effort
- Maintains only objects from the current user session in memory
- Database operates using client server technology.

RASCAL is an applications development tool based on the principles of O-O design. Traditional application pieces such as menus, data entry forms, reports, database cross-reference lookups, etc., are implemented as object classes. Every menu used in an application, for example, is described in the same manner using RASCAL. 4GL description is stored in an object database and retrieved dynamically at runtime as any other data would be. A higher level of object can then be created by combining these lower level objects. The key resources within an organization, such as parts inventory in a manufacturing company, are defined as an individual resource object, and all of the traditional pieces of an application are assembled by this resource object. This powerful feature allows the developer to call upon a resource as an individual object in order to implement all processing.

In order to reduce programming effort, RASCAL provides predefined ways of handling data:

- *Resources.* These are usually represented by relatively static data that describes things such as addresses, names, and other characteristics. This information is used on a day-to-day basis to support some kind of activity.
- *Sets.* Ordinarily, the objective in putting together a software system is typically to record this real-world activity. RASCAL allows you to model this through the use of sets, which are simply lists of information recorded under a common heading. For example: accounts payable is a series of data records, all under the heading of a common customer.
- *Phased Transformation.* The most powerful modeling option that RASCAL provides is the phased transformation that allows you to model a series of transactions with very little programming effort.
- *Ledger.* A final step in building many real-life systems is to generate entries in a general ledger summarizing the financial activity that has taken place. For this purpose, RASCAL provides special ledger-handling facilities.

RASCAL will run on DOS, Unix/Xenix, VMS, AIX.

Hardware: Intel 80X86, HP9000, SPARC, VAX, Amdahl, IBM 6000.

Pricing: from $325 up.

Product can be purchased from the company directly. Full training facilities are available in Corporate Office as well as at your own office.

Product News

IntelliCorp's KAPPA — High-Performance PC Software Development Environment

Ruth Hobler
Franson & Associates
(408) 453-5220
Erin Barrite
IntelliCorp
(415) 965-5639

IntelliCorp, Inc., announced **KAPPA,** a new high-performance software development environment for personal computers that combines the technologies vital to the development of low-cost, high-impact business applications.

Based on the C programming language, KAPPA features powerful object-oriented programming, high-performance rule-based reasoning, active graphics, and intelligent links to other applications, spreadsheets, and databases. Together, these features let software developers build and deliver solutions for their most important business operations quickly and easily, on standard platforms.

"IntelliCorp is the only software tools company delivering the environment that businesses and independent developers will need for the next generation of applications," said Skip Hawk, IntelliCorp's senior vice president of sales and marketing.

The first release of KAPPA, scheduled for the end of April, runs on an IBM AT or compatible PC with 640K of RAM, the MS DOS operating system and the MS Windows 2.1environment. The US list price of KAPPA is $3,500; runtime versions, which can be embedded in the final applications, are $450. Quantity discounts are available. KAPPA is supported by telephone hotline, and a variety of extended maintenance options are available.

IntelliCorp is best known for its LISP-based KEE product line, industry-leading tools and services targeted at diagnostics, scheduling, design, planning, and configuration applications. The addition of the C-based KAPPA line broadens the IntelliCorp product offerings into more general-purpose application development. The company's customers span all industries, including financial services, automotive, aerospace, telecommunications, chemical, government, energy, manufacturing, and engineering.

IntelliCorp products are marketed through a combination of the company's direct sales force and its authorized dealers, resellers, and VARs.

Headquartered in Mountain View, California, IntelliCorp (NASDAQ: INAI) has 225 employees, with offices throughout the US, Europe, and Asia.

CX Hyper Kit

Computas Expert Systems
Einar Dehli
Vertiasveien 1
P.O. Box 410
N-1322 Hevik
Norway
V: 47 2 47 78 22
F: 47 2 47 98 71

Computas Expert Systems has developed a full-fledged hypercard/hypertext system in Smalltalk/V 286 and Smalltalk/V PM, with all the flexibility that only Smalltalk provides.

The package was originally developed for in-house use to speed up application development, but it proved so successful that steps have now been taken to turn it into a new CX Smalltalk/V extension kit, called the CX Hyper Kit. The positive reception of the CX Multiprocessing Extension Kit (CX-MEK), which started shipping at the end of 1989, has contributed to this decision.

Features include most standard hypercard facilities, plus:

- ➪ full inheritance between cards
- ➪ icon library with buttons, etc.
- ➪ mixed color text panes
- ➪ graph layout support
- ➪ text and graphics can be stored on disk or in memory
- ➪ various browsers, editors, and inspectors for interactive use
- ➪ full file out/file in support for offline editing
- ➪ compatibility with CX-MEK.

The CX Hyper Kit is not only for building applications with hypercard/hypertext capabilities; it is equally well suited for quickly building professional-looking user interfaces to almost any kind of Smalltalk/V applications.

Project Management/
Adele Goldberg and Kenny Rubin

Talking to Project Managers —
Case Studies in Prototyping, Part 2

This article is the fifth in a series initiated with the February issue of the Hotline on Object-Oriented Technology. In earlier columns, we introduced object-oriented technology as a potential solution to the problems posed by the "software crisis." In doing so, we focused on prototyping and software reuse as they are supported by the object-oriented approach.

This installment proposes structural changes in the software development team that should make it better suited to the use of object-oriented technology.

Members of a development team need to consider several organizational changes to benefit from object-oriented technology. In particular, development processes and task assignments for the team members change in consideration of:

- Prototyping as it affects the development life cycle;
- Managing growth and accessibility of the organization's library of reusable software components;
- Learning about and practicing object-oriented analysis and design techniques.

In addressing the topic of team structure, we have chosen to cover only those aspects resulting from the introduction and use of object-oriented technology. This article, therefore, does not cover several team issues, such as assignment of authority, process of decision making, and explicit and implicit channels of communication that exist among team members.

© 1990 ParcPlace Systems.

To be sure, these are problems of concern to any team but they do not necessarily manifest themselves differently when a team utilizes object-oriented software engineering as opposed to traditional software technology.

Furthermore, this article does not attempt to survey the team differences that exist due to project characteristics such as difficulty of task, anticipated amount of code in resultant software product, and duration of project. A vast body of literature exists that deals with differences between research and product-oriented projects, and identifies differences in team structure based on the assignment of decision-making authority (e.g., Weinberg's egoless team versus Baker's chief-programmer team). This article deals with product-oriented projects only, although many of the core ideas apply to research projects as well.

Object-Oriented Teams

In addition, this article assumes that the project is large enough to require multiple team member roles. The differences we identify apply to small teams, and are intended to scale for larger teams. We focus on team roles. It is possible that a single person takes on multiple roles, although highly unlikely that the quality control intended in separating analysis from design, and coding from testing, easily supports the same person in such a pair of roles.

We find it useful to approach object-oriented development projects with two distinct project teams: the *Application Team* and the *Framework Team.*

The Application Team is responsible for the design, development, and delivery of a domain-specific application that fulfills a contractual obligation with a client. This team specifically interacts with the client to determine the requirements, and focuses on reusing existing components in the course of development to meet deadlines in a timely way.

The Framework Team has a fundamentally different charter than the Application Team. The Framework Team is concerned with the development of a basic general framework that can be customized to obtain different end-user applications. Each of these applications is related in that each fits into a single domain of discourse.

Customization to obtain each of these applications means that optional existing components are selected by the programmer and then attached to the framework. In addition, the programmer might refine existing components, or create new ones. An integral part of an application framework, then, is the library of components that can interact in that framework.

Part of the motivation for a two-team approach is based on the observation that corporations develop a nucleus of expertise in a particular domain (e.g., radar systems, process control, transaction processing). Using this concentrated expertise, these organizations specialize in the development of related applications within a given domain.

The idea of a Framework Team is to focus attention on the design and development of a general framework that can be reused by a

number of different Application Teams to develop applications within the same general domain.

An example framework is a bank information system that consists of objects for managing customer accounts, objects for representing transactions, and various ways in which to create reports. Optional components for the bank information system include various kinds of accounts and templates for reports. A particular bank is described by specifying accounts that will be supported and reports that must be generated. The bank can be further described by creating new kinds of accounts or report templates.

It is important to understand that the Application and Framework Teams have a symbiotic relationship that drives them to achieve the same overall goal: to deliver a reusable, extensible, and maintainable system, within budget and on schedule, that meets the needs of end users. What distinguishes the teams is the path each uses to achieve this goal and the subgoals that are passed along the way. In the remainder of this article, we describe the members of each of these two teams.

Object-Oriented Team Members

In our earlier articles on the inclusion of prototyping in the software development process, we suggested that the life cycle consists of the following activity categories:

- *Analysis Prototyping.* The purpose of producing the Analysis Prototype is to obtain the formal requirements specification.
- *Design Prototyping.* The Design Prototype demonstrates the feasibility of a particular implementation architecture.
- *Reuse Testing.* The components are tested to determine whether they can be reused to create a variety of applications based on the application framework.
- *Reuse Assessment.* The application framework and components are tested to determine whether they can be reused for purposes outside the current project, and whether they are sufficiently general to warrant inclusion in the corporation's library.
- *Product Implementation.* The product evolves from the Design Prototype, engineered to complete all functional requirements.
- *Product Testing.* Test suites exercise the prototypes and the product to determine whether all functionality works as specified.

These categories differ from the more traditional ones in that they incorporate prototyping and reusability. Tools that support object-oriented analysis, design, and implementation must be identified and supplied to the team members who work in each of the categories.

Examining the roles of development team members in terms of the six categories, we identify several roles that differ from those in a traditional development team. For the most part, existing team members can be retrained to fill these roles. The Application Team includes:

- *Analysis Prototyper*
- *Design Prototyper*
- *Reuse Tester*
- *Reuse Librarian.*

The *Analysis Prototyper* works with the end user to create an executable version of the proposed product. This person must have good interviewing skills to accurately solicit relevant end-user requirements. Often, the end user does not know the exact requirements; in such cases, the Analysis Prototyper and the end user must work together in a participatory manner to discover them.

The Analysis Prototyper need not be a programmer if 4GL-like tools for the object-oriented prototyping system are available. For example, a visual-interface construction kit would be useful in constructing a mock-up of the desired application and in providing the end user with an executable proof-of-concept. The goal of the Analysis Prototype is an engineering requirements specification. If the final specification is to be prepared by the Analysis Prototyper, rather than a documentor, then the Prototyper should have professional writing skills as well.

It is important that the Prototyper, who must actually create the prototype, be able to ignore details such as performance, architectural quality, and size, as these are not essential during this phase of development.

The *Design Prototyper* is a conceptual thinker, one who understands the notion (dynamics and mechanics) of an architectural framework. Furthermore, although this Prototyper understands the details of implementation requirements, he or she can overcome the desire to polish the prototype code, and instead can focus on how to organize requirements into conceptual categories. The Design Prototyper is an excellent programmer who can create the Design Prototype, as well as an excellent explainer who can prepare the architectural specifications.

The *Analysis Prototyper* is evaluated on timeliness of preparing the requirements specification to the satisfaction of the end users involved. He or she is also evaluated on whether the Design Prototyper can translate the specification into an architectural proposal. The Design Prototyper is evaluated on the ease with which the design is translated to implementation, the flexibility with which the design can be modified, and the extent to which the design allows the implementor to reuse components from the corporation's repository. Design Prototypers are evaluated on their contributions to the repository, whereas coders are evaluated on their withdrawals from that library.

For both kinds of prototypers, it can be useful to have a separate person act as Documentor. The Documentor is then responsible for

nology. DG's DAA takes the object model and extends it across all machines in the network. Users can access facilities anywhere in the network using the same graphical interface that is used for their personal productivity desktop applications. ...The focus of distributed object management is to provide object-oriented implementation and management of network based resources and services.

Source: "The Workstation Style," DG REVIEW, *Sanford Friedman, March 1990, pp. 28,31–34.*

Hewlett-Packard's efforts to get NewWave accepted as an industry standard have been given a boost from an accord between the Object Management Group (OMG) and X/Open. The X/Open agreement adds validity to HP's efforts to get the Object Management Facility (OMF), a vital part of New Wave's object-oriented environment, accepted as an industry standard.

Source: "HP's OMF Effort Gets Boost From OMG., X/Open," THE HP CHRONICLE, *Lane Boyd, March 1990.*

Hewlett-Packard Co.'s NewWave office automation software continues to gain widespread industry support, with AT&T and Lotus Development Corp. announcing support for the object-oriented user interface last week. ...joining NCR Corp., Data General Corp., and Cannon.

Source: "NewWave is Backed by AT&T and Lotus," MIS WEEK, *Tom Quinlan, March 19, 1990.*

Decisis Inc. a new subsidiary of Xerox Corp., introduced a decision support program last week that uses object-oriented programming structures and artificial intelligence techniques, offering users a powerful toolkit for solving problems.

Source: "New Xerox Subsidiary, Decision Support Tool," COMPUTER RESELLER, *Philip Rose, March 12, 1990, pp. 93,94.*

Standards

...vendors are working to create a standardized version of the [OOP Pascal] language to let developers port both their code and their knowledge from compiler to compiler. The Joint Pascal Committee, a standards-development body for ANSI that comprises compiler vendors and other industry experts, is seeking to develop a standard OOP Pascal. ...Some observers express doubt that the compiler companies will be willing compromise their proprietary interests in favor of a unified ANSI standard. ...Rather than approving any of the current OOP Pascal derivations as a standard, the ANSI committee is likely to combine parts of the current OOP Pascal extensions with new extensions, while maintaining compatibility with the most recent standard for procedural Pascal, committee members said.

Source: "Pascal Group Aims to Set OOP Standard," PC WEEK, *Paul M. Sherer, March 12, 1990, pp. 53,55.*

The Object-Oriented Cobol Task Group, formed to propose standards for object-oriented extensions to one of the oldest computer languages, has issued a call for members. ...Current members include Micro Focus Ltd., Realia Inc., HP, IBM, Sun Microsystems, Wang Laboratories, Digital Equipment Corp., Interactive Software Engineering Inc., SPS Systems U.S.A., Hewitt Associates, and Allinson Ross Corp.

Source: "OOP Cobol Group Plants to Set Language Standards," INFOWORLD, *Stuart Johnston, March 26, 1990.*

Testimonials

Turbo C++ is the best development environment we've seen for C++ programming. This is "industrial strength" OOP with a complete implementation of the latest AT&T specification for C++.

DHL services large customers with heavy shipping requirements by providing turnkey systems for onsite use. In addition to the main application, several utility programs provide interfaces to other DHL computer systems. Our group works with multiple source code files simultaneously (including files larger than 64K in size), and manages the utilities using Turbo C++ project management.

Currently we are rewriting a C utility that uploads billing information to our host systems. We'll exploit Turbo C++ OOP features to reduce our code size and improve performance. Once this project is done, we'll use Turbo C++ to build our own class libraries, taking advantage of library enhancements such as new video functions, and rewrite our core application.

Source: DHL Worldwide Express, *Peter Blair, Global MIS Department.*

We explored Turbo C++ object-orientation fully, creating objects that would handle both asynchronous or network communications with the same code. By constructing a hierarchy of classes, variations between asynchronous and TCP/IP network communications were automatically derived. The implementation of AT&T's 2.0 specification is solid as a rock.

We wrote our network monitoring program initially as a single session with one common object. Then we added support for multiple sessions using the "new" command defined in AT&T's specification, without rewriting any of the original code.

Borland's Virtual Realtime Object-Oriented Memory Management (VROOMM) reduced our memory requirements by 25% while preserving program performance levels. Much of the memory-intensive initialization and termination code normally required in programs is automatically placed in an overlay that can be used once and thrown away. This represents a tremendous breakthrough for networking environments where protecting memory resources continues to be a challenge.

Source: LAN Software Development & Management, *Todd Booth, President*

are one answer to today's systems development challenge. ...Large system development efforts are switching to the object-oriented view of the world, because systems that are too complex to understand in the whole can be modeled with O-O methods, masking the complexity to get to an understandable level. Computer-aided software engineering (CASE) vendors are rewriting their commercially available products using object-oriented techniques and languages in order to lower overall software development costs, allow rapid delivery functions, and to be more responsive to rapid changes in the marketplace.

Source: "A Hardware Approach to Software Reusable Parts," SYSTEM BUILDER, *Dennis Farley and Tony Scott, January 1990, pp. 44–46.*

Few, if any, of today's CASE tools facilitate the design and implementation of object-oriented systems. This deficiency will need to be remedied.

Source: "Technology and the DP Shop: Designware Changes Software Development," SYSTEM BUILDER, *J. Terence Gallagher, January 1990, pp. 20–21.*

...authoring software has made major inroads over the past few years, leading to today's object-oriented programming. With OOP, a person can think the steps of a problem through and program them by means of picture icons, even if their knowledge of programming is elementary... This is not to say, however, that programmers are no longer needed. Indeed, there will always be a demand for good programmers, especially in the systems area, people who can write the sophisticated packages that will enable casual users to become "power" users.

Source: "Is the Programmer Obsolete?," COMPUTING CANADA, *Patrick Abtan and Jean Julien, March 15, 1990.*

Mike McCoy, assistant dean for Medical Informatics at UCLA's School of Medicine, has been using OOP techniques to develop software for over four years. He reports that it took one month for a developer with no OOP experience to complete a simple report-generator project that used five objects. According to McCoy, the project would have taken at least three months to complete using more traditional methodology. [Brian] Barry [project leader for real-time embedded systems at the Canadian Defense Research Establishment] estimates that it takes longer than that to become really comfortable with OOP. It took three to six months for his developers to become productive. ...One main problem in trying to measure the effectiveness of OOP is that there is no standard measure of productivity.

Source: "Object-Oriented Programming: Will it Work for Real-time Systems?," COMPUTER DESIGN, *William R. Ford, March 1, 1990, pp. 44, 46.*

Much current comment confuses object-oriented programming with visual programming, because object-oriented approaches are so handy for manipulating graphics.

Source: "Mix Your Metaphors: Trade-Offs In Visual Programming," RELEASE 1.0, *March 16, 1990.*

I [author, R.A. Baker] would really like to see a short piece or get suggestions that would convince me that object-oriented programming (OOP) or object-oriented databases (OODB) do anything beyond convince management to be patient. ...the proponents have done a marvelous job of selling the concept... but can they give me one finite...example of how C++ or its kin can do better than a well-defined library in Fortran or C?

Source: "Show Him the Way," INFORMATION WEEK, *R.A. Baker, March 19, 1990.*

In 1967, two Norwegian computer scientists, Drs. Dahl and Nygaard, adopted a new unique approach to simulation... Two years later, this object-oriented thinking influenced the research of Alan Kay, one of the prime movers behind the Apple Computer and Smalltalk. Since then, literally dozens of O-O programming languages (OOPLs) have been produced. Scores of O-O database management systems (OODBMSs) are being researched and developed in companies and universities throughout the world. Object-orientation has many applications — just one of which involves DP software. Other O-O applications have included areas such as chip design, robotics, machine control units, and even manual organizational methods.

Source: "Object-Orientation and its Software Implementation," DATA BASE NEWSLETTER, *James J. Odell, April 1990, pp. 1, 5–8.*

Companies Supporting Object-Oriented Technology

Ah, the changing face of Chicago ...object-oriented programming...is not yet widely known throughout the Windy City, but if a number of organizations here have anything to say about it, it is just a matter of time. Two groups in particular, the Information Industry Council, based here [Chicago], and the Technology Innovation Center (TIC), based in Evanston, Ill., are looking to stimulate the growth of high-technology business in the area. "The two groups are working to perpetuate the development of technology in the area..." said Jim Currie, director of business development for the TIC. The [Information Industry] Council is supported by many of the large information and technology corporations that have headquarters based in Chicago, or a strong regional presence. These include Apple Computer Inc., IBM Corp., Motorola Corp., and AT&T Co.

Source: "Area Groups Push Software Development: AT&T, IBM and Apple are Among Supporters," COMPUTER RESELLER, *Mary Lee Shalvoy, March 26, 1990.*

Major vendors have all announced plans for network-wide computing. ...Data General (DG) has announced DAA, its Distributed Applications Architecture, which is based on object management tech-

maintaining readable, understandable, and accessible documentation of the current on-line system. This is an engineering position.

Reuse testing is a very special approach to testing, since it determines whether or not software components have been designed for broader applicability. Designing for reuse is a key way in which object-oriented technology is leveraged within the project and, later, across projects. Whereas the Design Prototyper is evaluated on an ability to create reusable components, it is important to provide special testing of these components. Normal testing covers, of course, whether the components work according to specification. But the Reuse Tester brings added value by discerning potential new uses for the components.

The Reuse Tester, then, works with the Design Prototyper to recommend new entries in the repository of reusable components.

The *Reuse Librarian* is responsible for classifying, documenting, and entering new reusable components into the corporate or project repository. The contents of the repository have to be carefully controlled so that only components that have passed special review are included, and so that once components are added to the repository, the external interfaces are consistently maintained. Documentation for each component should include indications of the context for which the component was designed, ways in which the component has already been reused, and a history of modifications.

The Reuse Librarian is the most significant addition to the development team. He or she must be a skilled programmer who can evaluate proposed entries and recognize when new entries are actually refinements or generalizations of existing entries. As part of the development life cycle, design reviews are conducted at each phase of work. The Librarian should participate in every review to identify where reuse is not being properly leveraged, and to identify opportunities for adding reusable components to the repository. In addition, it is the responsibility of the Librarian to obtain reusable components from commercial vendors when appropriate.

In larger organizations, it may be useful to have a Corporate Librarian, who is responsible for the corporation-wide repository of reusable components, and multiple Project Librarians, each of whom maintains the new components generated by an individual project.

Within such an organization, project members are free to use any components in the corporation-wide repository, as well as any components in their own project repository. They are not, however, permitted to use components from other project repositories. Doing so can lead to the propagation of unreviewed and undesirable code into Design Prototypes and products.

In general, until a formal review has taken place, the corporation should assume that components in project repositories may not be ready for corporation-wide use. Such reviews normally take place upon completion of a project, where the Corporate Librarian works with an individual Project Librarian to evaluate project-generated components that might be added to the corporate repository.

Many people have told us that the role of Librarian would not be a coveted position in their organizations. We counter these comments by noting that compensation for the Librarian should reflect the high value that the corporation must place on the Librarian's contributions to corporate productivity. In addition, we propose that the Librarian participate in each project's design review to make sure that reusable software assets are actually being reused.

When the team goal is to create an application framework, the team assignments are modified to take into account the creation and testing of both the application framework and library of reusable components. The Framework Team consists of:

- *Domain Analyst*, responsible for providing a detailed analysis of a given domain of discourse, such as Banking Systems, Radar Systems, Expert Systems, Reporting Forms, or Manufacturing Control.

> **"...few supposedly knowledgeable speakers and teachers ...emphasize the behavioral viewpoint"**

- *Designer/Implementor*, responsible for taking the Domain Analysis result and designing a reusable application framework.
- *Framework Tester*, responsible for testing the reusability of the designer's framework. This requires the tester to develop typical as well as atypical usage scenarios to evaluate the reuse effectiveness of the framework.
- *Components Tester*, responsible for testing whether the components interact correctly with the existing framework and whether the components are independently reusable for other domains.
- *Librarian*, responsible for classifying, documenting, and entering new reusable components into the corporate or project repository, and for recognizing reuse opportunities in design meetings.

With the exception of the Domain Analyst, all of these roles must be filled by competent programmers. The intention here is

that a different person will be responsible for each role. Moreover, the corporation's library itself can be evaluated on how well it supported the prototyping process.

Educating Personnel about Object-Oriented Technology

Structuring the development team in the manner we have identified is important to the successful use of object-oriented technology. But this success presupposes that the team members are knowledgeable about how to use the methodologies and programming techniques that embody the technology.

Usually, the additional team members we have identified do not already exist. Training is required to learn the skills of object-oriented analysis, design, and coding. In this section, we briefly review object-oriented analysis and design topics that should be learned by the various team members.

Object-oriented analysis is about identifying behaviors and assigning these behaviors as the responsibilities of different entities. Each entity thereby plays a role in the application by contributing to the full set of required behaviors.

Taking a behavioral approach is the hardest change, but essential to the successful leveraging of the technology. Nonetheless, we have been surprised by how few supposedly knowledgeable speakers and teachers on the object-oriented approach emphasize the behavioral viewpoint. Whenever we teach analysis in our methodology seminars, we spend most of our time teaching observation skills — how to look for and categorize observed behaviors. These observed behaviors are then associated with entities that exhibit the behaviors and/or carry out the responsibilities for each behavior.

We have encouraging experience that high-level managers, both technical and nontechnical, can understand behavioral analysis and how it provides a classification of desired system requirements. Object-oriented analysis enables a direct mapping from the observable and desirable behaviors to a classification scheme that drives the conceptual layer of an object-oriented architecture. As we said earlier, well rounded analyzers must be good interviewers, interviewing those people knowledgeable about the desired outcomes. And good analyzers must be good conceptual thinkers as well.

The skilled object-oriented designer is also adept at classification techniques. Classification involves grouping objects that behave similarly and have similar kinds of visible properties. The life cycle of these objects must also be taken into account to understand the succession of states that the properties and behaviors display.

Object-oriented design involves grouping the identified objects according to a number of possible relationships, such as ownership, specialization, delegation, and dependency. Each object relationship needs to be well understood by the object-oriented designer and then the implementor needs to be able to map the specification of each relationship to the available language constructs.

Teachers of object-oriented design, then, must take the goals of the design and map them to general design activities, and then link these activities to specific object-oriented mechanisms. There are five design goals attainable with object-oriented technology: reducing complexity, identifying parts, and delegating responsibility, capturing discovered commonality, sharing behavior across otherwise unrelated entities, and relating entities through process control. All the programmers on the development team should understand the fundamentals of object-oriented design, and be able to map the various design goals into the mechanisms supported by object-oriented systems.

There are many courses appearing, faster than might be desirable in that it is difficult to determine where the really skilled teachers exist. We believe a first filter on which courses to take should be whether the teachers are themselves experienced practitioners. It is also important to discriminate between specific language/system courses and broader methodology seminars.

Team managers, documentors, and analysts, as well as all programmers, should take the methodology courses, whereas only the programmers require the language/system courses. All participants in the corporation who take the same methodology courses will have a shared vocabulary for discussing product concepts.

Apprentice Programs have recently been forwarded as an effective approach to learning. The idea is to place key members of a team at the site of an active group of contract consultants who can monitor early development experience. In a later issue, we will discuss the effectiveness of Apprentice Programs, and provide more details about what needs to be covered in any course on object-oriented analysis, design, and implementation. ■

Dr. Adele Goldberg is President and CEO of ParcPlace Systems. Adele spent 14 years as a researcher and manager at Xerox Palo Alto Research Center. She has served as president of the ACM Systems Software Award.

Kenny Rubin is a computer scientist at ParcPlace systems, where he has been a primary curriculum developer of three of the ParcPlace Systems Smalltalk-80 training courses. In addition, he has served as lead instructor for many of the ParcPlace Systems courses taught over the past two years and has written on object-oriented technology, artificial intelligence and human-computer interaction.

Both authors may be reached at ParcPlace Systems, Inc., 1550 Plymouth Street, Mountain View, CA 94043; (415) 691-6700.

Publicity Corner

Clippings Summary

The number of clippings totaled 124: 39 articles in the weeklies, 23 in the biweeklies, 60 in the monthlies, and 2 in the dailies.

About 20% were articles covering object-oriented programming, languages, and tools; 22% were object-oriented hype or marketing news; 42% were product announcements or reviews; and 9% were general technical articles. The remainder (7%) were made up of editorials and graphic applications using object-oriented technology.

Applications

AutoCAD and NC-AUTO-CODE were used to design, evaluate, and program every part of the 900-pound [ice] drag racer, according to [Gary] Beyer and [Keith] Billet [of Billet Tool, manufacturer of all custom machined parts for the ice dragster]. AutoCAD allowed quick determination of the weight of each part before it was produced. Design changes, forced by unavailability of originally specified parts, were a way of life for a while. Fortunately, reprogramming was easily accommodated with NC-AUTO-CODE's object-oriented capabilities.

..."We didn't have time to write programs and spend hours debugging them. Using NC-AUTO-CODE, we can write programs, see the tool path on the part while we are programming, send the code to the mill, do a quick dry run, and go to production, all in the same day," Beyer said. "Usually, I don't even look at the code before I send it to the controller, the system is that good. Programmers are just the way everyone in the shop wants to see them. It's the only way to run a job shop in today's environment where every dollar counts."

Source: "NC-AUTO-CODE Goes Racing," CADENCE, *Bill Shields, Feb. 1990, pp. 51–54.*

The numbers are huge. This year alone, U.S. companies will spend $7 billion on 3 million tons of forms... and paper is supposed to be on the way out in the business world. Because of this, a number of companies are seeking ways to automate forms processing. ...Xerox Desktop Software has come out with FormBase 1.0, which the company calls "the first software package to fully integrate professional forms-creation capabilities with an object-oriented database manager." What this means is that it will help you create almost any kind of form for your business, manipulate the data to be entered in the form, and print the form. And if either the form or the data changes, you don't have to restructure the form from scratch.

Source: "FormBase: Help Automating Business Forms," COMPUTER SHOPPER, *Suketo Mehta, March 1990, pp. 310, 318–319.*

Robot Independent Programming Environment (RIPE) software development environment, designed by computer scientists David J. Miller and R. Charleene Lennox at Sandia Laboratories, improves programmer productivity, speeds the process, and lowers the cost of bringing new robot systems into use. RIPE has been successfully applied to widely differing intelligent robot systems.

...RIPE is designed with object-oriented programming. ...With this interface, a user can change one robot for another without having to reprogram the application. The application software remains the same because the interface is the same no matter what type of robot is being used.

Source: "RIPE Simplifies Programming of Smart Robots," ADVANCED MANUFACTURING TECHNOLOGY, *March 1990.*

RIPE programming environment developed at Sandia not only speeds programming of robots and other flexible manufacturing components, but it also cuts programming costs. Economics of these systems are frequently dominated by the cost of software development. Object-oriented programming is central to RIPE Design.

Source: "Flexible Manufacturing Software Writing Speeded," INSIDE R&D, *March 14, 1990.*

As software costs and time-to-market pressures continue to grow, ...that philosophy [of skepticism toward switching to OOP] may have to be discarded. ...Patrick Ferrel, a senior software engineer at Aldus claims that OOP has saved significant amounts of time in developing an upgrade of the company's pagemaker software. ..."Unless they switched to an OOP methodology, it would be impossible to meet schedules," Ferrel says. ...When Release 4.0 was being developed, about 20% of the code was written in C++ using OOP, which speeded the project considerably, according to Ferrel.

Source: "Object-Oriented Programming: Will it Work for Real-Time Systems?," COMPUTER DESIGN, *William R. Ford, March 1, 1990, pp. 44, 46.*

Newsworthy Revelations

Today's software systems require ways to mask complexity, model real-world components accurately, and, once constructed, remain flexible enough to accommodate change rapidly. Software reuse techniques

Company Showcase

Oregon Software

For more information call 1(800)874-8501

Portland, Oregon – April 30, 1990. **EMELELGY,** an investment group, purchased **Oregon Software, Inc.** today. This takeover establishes a sound financial footing for the company and gives Oregon Software the opportunity to develop new markets. There are no plans to change the organization of the Portland, Oregon software company.

The new owners plan to utilize Oregon Software's technical leadership to strengthen the company's position in the emerging object-oriented programming market. Object-oriented programming is the leading growth area in the computer industry. Robert Martin, President and CEO, believes Oregon Software is in a better position to provide C++ development solutions in the object-oriented programming market than any of its competitors. Oregon Software introduced the first Pascal compiler for DEC minicomputers in the late 1970's and the first true C++ compiler in mid-1988.

Oregon Software currently offers an optimizing C++ development system containing a compiler that generates fast, compact object code and a full-featured source-level debugger. Furthermore, Oregon C++ is fully compatible with AT&T's version 2.0 standard for C++ and is switch-selectable for ANSI C and K&R C. Oregon C++ runs on most Motorola and Intel UNIX platforms, as well as VAX/VMS and VAX/Ultrix. DECstation and SPARCstation releases are planned for this summer.

Oregon Software will continue to support it's Pascal and Modula-2 development systems.

Computas Expert Systems

Einar Dehli
Vertiasveien 1
P.O. Box 410
N-1322 Hevik, Norway

V: 47 2 47 78 22
F: 47 2 47 98 71

Computas Expert Systems A.S (CX) of Norway announced today the CX Multiprocessing Extension Kit for Smalltalk/V, with immediate availability forSmalltalk/V and Smalltalk/V 286. Versions for Smalltalk/V Mac and Smalltalk/V PM are scheduled for release later in 1990.

Computas is a pioneer in commercializing AI technology in Scandinavia. The company has developed its own object-oriented approach to knowledge-based reasoning, graphical user interfaces, and systems integration.

Smalltalk/V is the increasingly popular and powerful object-oriented programming environment from Digitalk, Inc., of Los Angeles, CA.

In recent years the Smalltalk/V family has been employed by CX in systems ranging from real-time energy management systems in steam plants to "electronic rulebooks."

Experience from these systems has put CX in a unique position to build high-quality Smalltalk/V extension kits that stand the tests of strict industrial requirements.

The first Smalltalk/V extension kit from CX, the CX Multiprocessing Extension Kit (CX-MEK), has been available since late 1989 in a version for Smalltalk/V and Smalltalk/V 286. CX-MEK helps build Smalltalk/V applications faster, whether you are interested in multiprocessing or not. CX-MEK provides more than 900K of industrial-strength Smalltalk/V source code, accompanied by a 116-page, high-quality manual. Versions for Smalltalk/V Mac and PM are under way.

The CX Multiprocessing Extension Kit, containing 15 modules totalling 900K of high-quality, reusable Smalltalk/V code, is a must for all Smalltalk/V programmers. In addition to giving high-level support to all kinds of multiprocessing applications, the kit includes many environment extensions that will be appreciated by all Smalltalk/V users. The multiprocessing extensions include a process status window, the ability to update partly occluded windows, event scheduling, process communication primitives, a data acquisition protocol, and much more.

The environment extensions include a form inspector, a snap window facility, enhancements to the built-in drawing package, many text-editing accelerators, text wrapping, menu-based access to previous versions of method sources, the ability to browse arbitrary large files in the disk browser, etc. (Some modules depend on other Digitalk Extension Kits.)

The combined Smalltalk/V and Smalltalk /V 286 version is favorably priced at $99.95, including postage and handling. Orders must be prepaid.

Analysis & Design/Edward V. Berard

Object-Oriented Requirements Analysis

Before I begin a discussion of object-oriented requirements analysis (OORA), I must stress several important points.

First, as I have stated in several articles in the past, the development part of the object-oriented life cycle is best accomplished using a recursive/parallel approach, i.e., "analyze a little, design a little, implement a little, and test a little." Therefore, even though I may talk about OORA as if it were performed at only one place during development, in reality, it may very likely be accomplished at many places.

Second, although it may be preceded by such things as a feasibility study, I will treat OORA as if it were the very first thing to be accomplished during the development of a software product. For example, if you were to attempt an object-oriented approach using the classic waterfall life cycle model (not recommended, but still possible), OORA would be followed by object-oriented design (OOD), which would be followed by object-oriented programming (OOP), and so forth.

Third, OORA is not always appropriate. If the project is small and/or noncritical, OOP may be all that is required. As we move to larger efforts, OOD becomes a necessary precursor to OOP. For still larger and more critical projects, we often turn to OORA, followed by OOD, followed by OOP.

Object-oriented requirements analysis (sometimes referred to as object-oriented analysis (OOA)) was virtually unheard of in the object-oriented programming community until very recently. There are two main reasons why it is now being seriously considered:

- When object-oriented technology is applied to large and/or critical projects, the bottom-up approach so common in OOP often proves insufficient. Very few people, for example, can contemplate a project of, say, 100,000 lines of code, and adequately identify most of the low-level components without some form of analysis. As object-oriented thinking moves into the mainstream, people are realizing that OOP techniques (e.g., identify classes, create objects, and send messages) *alone* may be very inadequate for large, critical efforts.
- Object-oriented technology is now being seriously considered by people who are accustomed to thinking in terms of life cycle methodologies. These people include, for example, those developing large business applications, critical real-time applications, and large, complex software in general.

History

Although E.W. Dijkstra first describe what he called "structured programming" in 1969, it was not until December 1973 that many programmers became aware of the term. (The December 1973 issue of Datamation was devoted to the subject.) In 1974, the first article on "structured design" was published in the IBM Systems Journal, and in late 1976, Tom McCabe began publicizing something called "structured testing."

By 1976, there was some early work going on in the area of structured analysis. In 1977, Gane and Sarson were describing their version of structured analysis, which they later published in book form (*C. Gane and T. Sarson,* **Structured Systems Analysis: Tools and Techniques,** Prentice-Hall, Englewood Cliffs, NJ, 1979). However, it was not until Tom DeMarco published his best-selling book (*T. DeMarco,* **Structured Analysis and System Specification,** Yourdon Press, New York, NY, 1979) that structured analysis began to become generally accepted.

By 1983, it had become apparent that standard structured analysis was not adequate for real-time systems, e.g., it did very little in the area of such things as interrupts and scheduling of concurrent processes. A new, real-time flavor of structured analysis began to emerge. This real-time trend is also very apparent in modern OORA.

In an earlier article, I described how Grady Booch introduced the concept of *object-oriented design* to the Ada community. By 1986, OOD was being used on a significant percentage of Ada development efforts. However, there were some serious problems reported, e.g.:

- Traceability was difficult. (Traceability is a measure of the ease with which a concept, requirement, or idea may be traced from one point in the software life cycle to another point.) For example, contractors were often (if not exclusively) furnished with *functional* requirements and encouraged to develop *object-oriented* code. Tracing functional requirements to functional code was relatively easy since the localization (i.e., around functions) remained constant throughout the development effort. Changing localizations (i.e., from functional to object-

oriented) in the middle of development made tracing requirements (and, hence, acceptance testing) very difficult.

- Testing and integration became a nightmare. A very common practice was to divide a large effort into several (more manageable) *functional* pieces. Each of these functional pieces was given to a separate team that then designed and coded the piece in an "object-oriented" manner. Given that objects are *not* localized in a functional manner, this meant that the characteristics of a given object were often distributed unevenly among the functional pieces.

  What this meant was that each team had a very small probability of gaining a complete understanding of any particular object. For example, one team might see object X as having attributes A, B, and C, and another team would see object X as having attributes B, D, and F. These differences only became apparent when it came time to integrate the work of both teams, and the first team attempted to hand off their version of object X off to the second team. Since integration often occurs fairly late in development, this made the required changes difficult and expensive.

- Needless to say, there was also a great deal of duplicated effort, since each separate team often (re-)developed objects that were already in use by other teams.

- Software engineers found that working with two vastly different paradigms was difficult. For example, many of them had problems when attempting to "bridge the gap" between structured analysis and object-oriented design.

By 1986, it had become obvious that some form of object-oriented requirements analysis was needed. Most efforts to develop an OORA methodology used either classic structured analysis, a real-time version of structured analysis, or a combination of either of these approaches with entity-relationship diagrams, as a starting point. To someone with a strong background in conventional object-oriented programming, these approaches seem strangely out of place.

My staff and I began to investigate the problem. From our consulting and training efforts with other organizations, and from our own experience, we knew the following:

- Attempting to base an OORA methodology on conventional requirements analysis techniques was a mistake. Whether you referred to these approaches as "functional decomposition" or "event-driven," they did not localize information around objects, and they ignored virtually all aspects of object-oriented technology.

- We would have to supply some form of graphical techniques. A purely textual approach was undesirable. Quite frankly, most of the time, people relate better to pictures than they do to words. When we began, we did not know which graphics we would use, but one of our most important criteria was that the graphics be object-oriented, or directly support object-oriented thinking.

- We had to have some mechanism for creating a specification for objects of interest. Structured analysis, for example, has a "data dictionary." We thought that we should have an "object dictionary," but we were not sure how to describe the "entries," i.e., the objects of interest.

- Merely specifying objects of interest was not enough. We had to have some mechanism(s) for showing how these objects were related and how they would interact. It was a given that these techniques had to be graphical.

- Reusability was a key issue. We had evolved an OOD methodology that emphasized reusable objects, and planned to extend this thinking into the OORA methodology.

- Whatever methods we came up with had to accurately reflect object-oriented thinking.

- The methodology had to be programming language-independent to the highest degree possible. Developing a methodology that "reeked of Smalltalk," might not be all that applicable, for example, to projects that were going to be implemented in Eiffel, C++, CLOS, or Self.

- The methodology had to be:

  Pragmatic: Real people, working on real projects, under real constraints, had to be able to use the methodology.

  Quantifiable: As much as possible, the methodology had to be precisely defined so that it was repeatable, and so that viable alternatives could be evaluated.

  Widely applicable: The approach should be one that can be used with little or no modification on a complete spectrum of applications, e.g., from business, to scientific, to embedded real-time applications.

  Tailorable: Each organization (and, often, each project) may wish to emphasize different things. Some may want to delete items, others will want to add things, and still others may want to rearrange the overall process.

Over the past four years, the OORA process has grown and matured. Techniques, graphics, and documentation have all been modified based on experience. The OORA process continues to mature.

A Suggested OORA Methodology

As I mentioned earlier, there is more than one school of thought on how OORA might be accomplished, e.g., (P. Coad and E. Yourdon, **OOA — Object-Oriented Analysis,** Prentice-Hall, Englewood Cliffs, NJ, 1989 and S. Shlaer and S.J. Mellor, **Object-Oriented Systems Analysis: Modeling the World In Data,** Yourdon Press: Prentice-Hall, Englewood Cliffs, NJ, 1988). I share some ideas in common with other OORA methodologists, but both they and I would agree that what I am about to describe is significantly different from other OORA approaches.

I will list the steps to be accomplished as if they were in sequential order. However, in practice, some steps may be rearranged (within limits), and many may be accomplished in parallel, e.g., it is possible to be working on steps 1 and 9 at the same time.

The general OORA process is:

1. Identify the sources of requirements information.
2. Characterize the sources of requirements information.
3. Identify candidate objects.
4. Build object-oriented models of both the problem and the potential solution, as necessary.
5. Relocalize the information around the appropriate candidate objects.
6. Select, create, and verify documentation for candidate objects.
7. Assign the candidate objects to the appropriate section of the object-oriented requirements specification (OORS).
8. Develop and refine the Qualifications Section of the OORS.
9. Develop and refine the Precise and Concise System Description.

Now let's briefly look at each of these steps in slightly more detail.

There is nothing specifically object-oriented about the first two steps. Regardless of the approach to requirements analysis one chooses, they will have to be accomplished. Sources of requirements information can include: existing requirements documents, knowledgeable people, existing software (including prototypes), and standards documents.

Characterizing the sources of requirements information is actually a two-part process. You must characterize the source of the information, and you must also characterize the information provided by the source. The source may be characterized in terms of availability, authority, credibility, responsiveness, longevity, ease of access, and types of information provided. The information provided by the source may be characterized in terms of form (e.g., textual, graphical, machine-readable, and verbal), completeness, how current the information is, which aspects of the product it addresses, and understandability, among others.

Identification of candidate objects requires that we define what "objects" are. I most frequently deal with the following objects: classes, instances (of classes), metaclasses, subsystems, and systems of objects. Subsystems and systems of objects are large object-oriented entities. Metaclasses are classes whose instances themselves are classes.

Each object will have to be documented in some form. Classes, metaclasses, and, in rare cases, instances, are usually documented with an Object and Class Specification (OCS, pronounced "ox"). OCSs have five main sections:

- a precise and concise description
- a set of graphical representations showing both the static relationships and the dynamic behavior of the object
- a description of the required and suffered operations and their associated methods
- a description of the state information for the object
- any constants and exceptions exported by the object.

Object-oriented models of both the problem and solution can be fashioned from the following set of graphical techniques:

- semantic networks
- state transition diagrams
- Petri net graphs
- timing diagrams
- object-message diagrams
- graphics for subsystems
- graphics for systems of objects.

Each of these graphical models can have an accompanying set of engineering drawing notes, and should also be supported by the object dictionary. Models can also be created using automated techniques, e.g., Smalltalk, Trellis, Actor, and Prograph.

A natural human tendency is to (re-)localize everything we know about an item in one place. In the case of a candidate object, this will most often be a mixture of object-specific information and application-specific (i.e., context-specific) information. To ensure a high degree of reusability, we must take care to separate the characteristics of the object in isolation from how the object is used in the application at hand.

By this time, we should be able to examine our library of reusable objects and be able to select items that closely match our requirements. If we do not find a suitable match, we will have to create some new documentation (e.g., an OCS) for our candidate object.

If you insist on doing all of your object-oriented analysis at once, you will very likely be creating an OORS. The OORS is divided into two parts: an object-general section, and an application-specific section. Oversimplifying, reusable objects are placed into the object-general section, and application-specific objects, along with an object-oriented systems specification, are placed in the application-specific section.

At sometime during the OORA process, the OORA analysts must create an accurate description of the system that will be delivered to the client. This description will most likely be a combination of textual and graphical information. This description, along with cost and schedule estimates, legal documents, and other appropriate items belongs in the application-specific section of the OORS.

Edward V. Berard is founder of Berard Software Engineering, Inc. He may be reached at 18620 Mateney Road, Germantown, MD 20874; (301) 353-9652.

HOTLINE on
Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ ***Yes,*** I want to be plugged into the latest thinking and developments in object-oriented technology. Enter me immediately as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	**2 years (24 issues)**
☐ ~~$299~~ $249	☐ ~~$598~~ $498

outside the US — add $24 per year for air service.

☐ Check enclosed (Payable to: HOOT — US currency drawn on US bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card # _____ Exp. date _____
Signature _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____

Send Subscription Orders To: *HOOT, P.O. Box 3000, Dept. HOT, Denville NJ 07834*
For immediate service—call the Hotline *212-274-0640* ***or*** *Fax 212-274-0646*

DG90

HOTLINE on
Object-Oriented Technology
The International Newsletter

588 Broadway, Suite 604
New York, NY 10012
(212) 274-0640

Do you know of a colleague who might benefit from reading THE HOTLINE? If so, please fill out below and we will promptly send them a ***free copy with your compliments.***

Send issue to:
Name: _____
Title: _____
Company: _____
Address: _____
_____
City: _____
State: _____ Zip: _____

Compliments of:
Name: _____
Title: _____
Company: _____
Address: _____
_____
City: _____
State: _____ Zip: _____

DG90S

HOTLINE on
Object-Oriented Technology
The International Newsletter

July 1990 **Volume 1, Number 9**

Cognos and Eiffel: A cautionary tale

Burton Leathers

COGNOS INVESTED MORE THAN $10 MILLION OVER A two-year period in Eiffel-based product development. The product was a powerful, dictionary-based report writer that was to be the cornerstone of a new 4GL product set. This effort was abandoned for both technical and commercial reasons. A review of the experience shows that Eiffel delivers many of the expected benefits of OOP; however, errors on the part of both Cognos and Interactive Software Engineering (ISE), the Eiffel supplier, resulted in project failure. Hindsight tells us that the implementation of Eiffel was not and is not ready for use in large-scale development. Our errors masked this truth for longer than we should have allowed. It is possible to learn many lessons about how OOP can be used from the mistakes Cognos made.

A brief history

October 1986

- Cognos decided that object-oriented programming was the appropriate technology to use for the development of its future products.

continued on page 3...

SCOOP-East

Conference review ..... ***Jacob Stein***
Brad Cox lecture (pg. 15) ..... ***Doug Bennett***

HAVING BEEN ACTIVE in object-oriented technology for over seven years (but humble enough to blush when called a leader in the field), I previously treated commercial object-oriented conferences with a good deal of skepticism. I have had two major complaints. One, the instructors blow into town to make their presentations and then disappear. Two, aside from the names attached to them, it is quite often impossible to distinguish between tutorials and product presentations. All too often, speakers have taken tremendous liberty in pushing their products or services instead of educating us on the technology and its impact on our industries.

Given my skepticism, I did not take the glossy brochure describing SCOOP-East very seriously.

Statements like, "the entire OOP industry will come together to give you the most intensive, practical and technical coverage of pertinent O-O topics," and, "only at SCOOP-East can you count on meeting industry leaders," caused me to chuckle. On the other hand, having been invited to participate on a panel at SCOOP-Europe and not having attended previous SCOOPs, I thought that perhaps I should check it out. In addition, it might be a good opportunity to catch up with many "industry leaders" I hadn't seen since OOPSLA in New Orleans.

continued on page 14...

INSIDE...

From the Editor ***2***
A farewell ***3***
What's in an object anyway? ***4***
Languages & Environment ***8***
Impact of Windows 3.0 on the OOP market
Project Management ***11***
Introducing reuse into the development process
Publicity Corner ***16***
Product News ***18***
Calendar ***19***

FROM THE EDITOR/MARIE A. LENZI

Along with object orientation has come much hope and promise for advancements in the development process. However, in practice it has not all been productivity and reusability. In this month's feature, **Burton Leathers** of Cognos shares with us a not-so-rosy experience with object orientation. Burton analytically and objectively addresses problems with both the development product and the development process. After the high excitement and anticipation of object-oriented benefits, the article brings us back down to earth.

One of the problems experienced by Cognos was tools, or lack thereof. **Andy Novobilski**, creator of the Stepstone browser, discusses the productivity and educational benefits of development tools (specifically, the browser) generally associated with object-oriented development environments.

Object-oriented conferences are becoming more lively. SCOOP-East, held in late May at the Wang Institute, was a clear indication of this. **Jacob Stein** of Servio Logic reviews the conference, and **Doug Bennett** of Design Ways reviews **Brad Cox**'s invited guest lecture.

By some gift of the muse(?), GUIs and object orientation have become closely associated. While SCOOP-East was in progress, Microsoft formally announced the most publicized secret of the year. **Zack Urlocker** of The Whitewater Group discusses Windows 3.0's impact on object-oriented technology.

Too often I find that managers and end users alike are obsessed with the notion of reuse. *Introducing Reuse into the Development Process* is the next entry in the "Talking to Project Managers" series by **Adele Goldberg** and **Kenny Rubin** of ParcPlace Systems. They bring the issue down to earth by pointing out that it takes effort to get reusable artifacts and that the responsibility falls not only on the developers, but also equally on the managers.

Although I have previously been a contributor, editing a newsletter is a new experience for me. Jeff told me that these will be *interesting* times. Isn't there an ancient Chinese curse ... ?

As far as THE HOTLINE is concerned, I will continue to target management issues and concerns, and I hope to increase and diversify the truly useful management information. I feel that without full management and organizational cooperation and commitment, object-oriented technology will not be permitted to deliver on its promises.

I find controversy stimulating. It makes for interesting (there's that word again) reading. Not everything you see in THE HOTLINE will reflect my personal opinion or taste. You may take exception to some articles as well. Challenge them, be passionate, write intense "letters to the editor." I'll publish as many of them as possible (expletives deleted).

I am interested in your opinion of the publication: what you like and dislike, and, most importantly, ***why!*** Your comments and opinions are requested and welcome, as are suggestions about topics you feel need to be discussed.

In the months to come, you may notice additions and changes to the newsletter. We'll be working to more precisely target technical and middle MIS and business management. We at THE HOTLINE will continue to offer the level of articles we have offered in the past and will enhance the publication with articles that address organizational and nontechnical issues related to the installation, implementation, and growth of object-orientation.

Beginning in September, **Steve Weiss** will host a "Dear Object Doctor" type of column. Steve will address technical and nontechnical management questions. Send your questions to Steve c/o SIGS Publications, 588 Broadway, Suite 604, New York, NY, 10012.

Marie A. Lenzi
Editor

Advisory Board

Jim Anderson
Digitalk, Inc.

Tom Atwood
Object Design, Inc.

Grady Booch
Rational

Larry Constantine
Consultant

Cort DeVoe
IBM

Adele Goldberg
ParcPlace Systems

Bertrand Meyer
Interactive Software Engineering

Reed Phillips
Knowledge Systems Corp.

Trygve Reenskaug
Taskon A/S

Bjarne Stroustrup
AT&T Labs

Dave Thomas
Object Technology International

Zack Urlocker
The Whitewater Group

Eugene Wang
Borland International

Jeff McKenna
Founding Editor

Richard P. Friedman
Group Publisher

Margherita Monck
General Manager

Elisa Varian
Production Manager

Susan Culligan
Art Director

Melanie Younossi
Marketing Manager

Gregory Tsichlis
Circulation Manager

The Hotline on Object-Oriented Technology (ISSN #1044-4319) is published monthly by SIGS Publications Inc., 588 Broadway, New York, NY 10012 (212) 274-0640.

©Copyright 1990 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher. Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30.

POSTMASTER: Send address changes & subscription orders to HOOT, P.O. Box 3000, Dept HOT, Denville NJ 07834.
Submit editorial correspondence to Marie Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280.

CLASSIX

Empathy Incorporated
Att: Richard Mitchell
P.O. Box 632
Cambridge, MA 02142
617 / 787-3089

Empathy Incorporated has announced Classix, a fully supported library of reusable C++ software components. The product incorporates object-oriented design techniques and heuristics to provide reusable C++ classes covering abstractions in the area of data structure design, mathematical support objects, SmallTalk-like classes and mimics of the primitive data types. The product also includes a parameterization utility that enables the efficient reuse of Classix classes with different data types. Included in the library are the following:

DATA STRUCTURES		SMALLTALK-LIKE CLASSES
Array	SortedDoublyLinkedList	Object
AVL Tree	SortedDoublyLinkedRing	Association
BinarySearchTree	SortedLinkedList	Bag
BinaryTree	SortedLinkedRing	Collection
BoundedArray	Stack	Date
BTree	Table	Dictionary
DoubleLinkedList	MATHEMATICAL CLASSES	DoControl
DoublyLinkedRing	BitVector	OrdCollection
Graphs	Complex	SeqCollection
HashTable	Matrix	Set
LinkedList	Vector	SortCollection
LinkedRing	MISC. CLASSES	Time
PriorityQueue	Boolean	
Queue	Primitive Data Types	
	String	

This set of foundation classes offers liberal amounts of inheritance and containment, vital attributes of any strong object-oriented system. Because of this, the product has the potential to enhance the productivity of your developers, as well as help teach object-oriented design and development techniques.

Classix is offered in source code for use with Glockenspiel C++, Sun C++, GNU C++, Zortech C++, and MPW C++.

Pricing: from $295 / Ship date: June 30, 1990

Calendar

THE HOTLINE CALENDAR presents conferences and meetings that focus exclusively on object-oriented technology. It should be noted that many other other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to Marie Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280.

July 16–18, 1990: SCOOP-Europe — Imperial College, London, UK. Co-sponsored by *JOOP* and BUCEC.

Keynote-Adele Goldberg. Tutorials, lectures panels, product presentations, and exhibits.

Contact: Chantal Olsen, BUCEC, 72 Tyng Rd., Tyngsboro, MA 01879, (508) 649-9731.

July 31–Aug. 1, 1990: O-O Systems Symposium — Toronto, Canada.

Conference Chair: Larry Constantine. Tutorials, Lectures.

Contact: Digital Consulting, Inc., 6 Windsor St., Andover, MA 01810 (508)470-3870.

Sept. 14–15, 1990: Symposium on OOP Emphasizing Practical Applications — Marist College, Poughkeepsie, NY.

Keynote: Bertrand Meyer. Tutorial: Richard Wiener. Technical sessions.

Contact: James Ten Eyck, Marist College, Poughkeepsie, NY 12601, (914) 471-3240 Ext. 606.

Sept. 25– 28, 1990: C++ At Work '90 — Meadowlands Hilton, Secaucus, NJ. Cosponsored by THE C++ REPORT & BUCEC.

The International C++ Users Group Conference. Technical sessions, tutorials, panels, BOF & exhibits.

Contact: Bob Daniels, (508)649-9731.

October 3–5, 1990: OOP vs. AI — San Francisco, CA. Sponsored by International Association of Knowledge Engineers.

Keynote: Dr. Edward Feigenbaum. Theme: Identifying, developing, implementing, and integrating knowledge-based systems in the marketplace.

Contact: Joan Scaffidi, IAKE, Georgetown P.O. Box 25461, Washington DC 20007, (301)231-7826

October 21–25, 1990: OOPSLA '90 — Fifth Annual Conference on OOP Systems, Languages & Applications — Ottawa, Canada. Cosponsored by ACM & ECOOP.

Technical sessions, lectures, BOF, tutorials, panels, and exhibits.

Contact: ACM (212)869-7440; Conference Division.

Product News

ObjectGraphics

The Whitewater Group
1800 Ridge Avenue
Evanston, IL 60201
708/328-2800

ObjectGraphics is a high-level, object-oriented graphics library for the Actor development system. The product extends the power of Actor allowing you to develop sophisticated interactive graphical applications. It includes comprehensive, platform-independent support for most 2D graphics including diverse shapes, bitmaps, rich text, curves, coordinated spaces, and more. It also includes ObjectDraw, a drawing application that provides you with a wealth of sample ObjectGraphics code.

ObjectGraphics consists of rendering tools such as Pens, Brushes, TextPens, and GraphSpaces to control the appearance of graphic images; support for graphics such as bitmaps, shapes, and text; control over drawing attributes such as line styles, fill patterns, colors, fonts, and font styles; and control over drawing combinations.

When graphics libraries claim to be object-oriented, they generally mean you can manipulate visual object. ObjectGraphics provides a true object-oriented development environment. It allows you to develop graphical applications using a set of graphic objects rather than procedure calls. These objects supply graphic behavior you would otherwise have to implement yourself.

The product is designed to be portable. It provides platform filters that act as a transaction layer between the application code and the target platform, making the application code developed under one windowing environment fully portable to any other where the product is supported. The platform filters also provide for device independence. You define all drawing activities to a "world coordinate space" and the product maps those coordinates to the physical device, whether it's a window, a printer, a plotter, or a bitmap.

Included in the product is ObjectDraw, a drawing tool written in Actor and ObjectGraphics. The tool provides sample source code to help develop applications. You can use ObjectDraw classes and methods as examples or incorporate them directly into your applications. Additionally, graphics you create using ObjectDraw can be used in applications.

Currently available for Microsoft Windows; version for Presentation Manager is under development.

Pricing: $395 / Ship date: July 1990

Interactive Software Engineering

270 Storke Road
Suite 7
Goleta, CA 93117

Interactive Software Engineering announced that it has ported Eiffel software on SONY NEWS 3860 based on MIPS R3000 processor. Eiffel is an advanced object-oriented language, method, and environment designed for the production of high-quality software. Eiffel emphasizes reusability, extensibility, correctness, and robustness.

"We're excited about the availability of Eiffel on this new platform," said Jean-Marc Nerson, Interactive's VP of Engineering. "The responsiveness is incredibly fast. The shipment of the upcoming Version 2.3 of the Eiffel compiler will start simultaneously both in the United States and in Europe with Societe des Outils du Logiciel (Paris)." According to J.F. Coulon, Executive Manager of SONY Microsystems France, " The introduction of the Eiffel Language in our Software Catalog fits SONY's specific strategy: give the best tools and object-oriented language for today an the future."

The Eiffel package includes a complete set of libraries. The libraries constitute one of the major benefits of using Eiffel. Built of carefully crafted reusable components, they cover fundamental data structures, X-based graphics and multiple windowing, parsing, lexical analysis, nongraphical windowing, and other applications. For developers the libraries provide a solid basis from which they construct quality software.

Interactive is also very active in spreading the word about object-oriented programming; it has recently launched the Eiffel Institute of Technology which offers a full curriculum of courses aimed at programmers and managers on object-oriented programming and other advanced software engineering techniques.

- Work was begun on the development of an object-oriented programming language (OOPL) that would be used internally.

May 1988

- It was decided that the development of a proprietary OOPL was a distraction from the company's main effort.
- Commercially available OOPLs were reviewed, and Eiffel was selected.
- Development of a major new product was begun using Eiffel.

1988/1989

- Problems were encountered with the performance of both the Eiffel compiler and with the programs that were produced using it.
- Major revisions were made to Eiffel (2.1) to address both sets of performance problems.

July 1989

- A thorough review concluded that we could succeed only if we wrote an Eiffel compiler that addressed the performance issues.
- It was decided that the development of a proprietary Eiffel was a distraction from the company's main effort.
- Development work with Eiffel ceased.

When is Eiffel appropriate?

Given the list of problems we encountered, it must be asked whether Eiffel, as it now exists, is a usable language. It is, but not in all instances.

Many of the problems we encountered were a consequence of the scale of the programs we were producing. A good proportion of the program performance problem arose from the fact that the run-time size of programs increases as the square of the number of classes (a problem ISE partially addressed in release 2.2). The larger the program, the larger the problem. Also, we found that the problems with the compiler resulted in some measure simply from the size of our system. They would still be present in smaller systems, but the available literature (e.g., *A Development in Eiffel: Design and Implementation of an Eiffel Simulator,* JOOP 2:1, pp. 27–33) indicate that they wouldn't be the show stoppers they were for us.

Some of the difficulties we encountered were a function of the fact that we were producing a suite of related products for a variety of platforms. Meyer notes in §21.1 of *Object-Oriented Software Construction* that a more potent mechanism is needed for the storage of class-related data than is now used by the compiler. Our problems with variant handling confirmed this. Had we been doing a single product for a single target, many problems would simply have vanished. In brief: one product, OK; several products serially, maybe; several products concurrently, *not advisable.*

Not all programs require optimal performance. Many workstation-based, interactive programs can afford to be "slow" because the delay created by suboptimal performance is below the threshold of concern for the user. If the user is content and nothing else is being delayed, then a slower than best-case program is sufficient. In these instances, the performance problems we had with Eiffel should not be a deterrent to using it.

How could we resume use of Eiffel?

The size and performance of the programs we produced with Eiffel made them simply unworthy of serious consideration. After detailed examination of

continued on page 6 ...

A farewell ... — *Jeff McKenna*

If you look at the masthead in this month's issue, you will see a major change — there's a new name over "Editor" of The Hotline. We welcome ***Marie Lenzi*** *on board as the new editor.*

Over a year ago, I was approached by SIGS Publications about a new monthly publication targeted to management, which would focus on the emerging object-oriented technology. What resulted from those initial discussions is The Hotline as you see it today. I am proud of what we accomplished: a timely and informative newsletter that has met its objective of presenting the major issues in the object-oriented field.

The Hotline has mirrored the tremendous growth in the field, with more and more information to be filtered and presented to you, the reader. My business, the McKenna Consulting Group, has also mirrored this growth, with a corresponding increase of demands on my time. Hence, I have passed on the Editorship of The Hotline to Marie. She brings with her new energy and an expanded vision for the future. I know that she will do a yeoman's (yeowoman's??) job in her new role. I wish her the best.

The Hotline would not have been possible without the considerable efforts of the writers, and I thank each and every one of them. They were most generous with their time and responded positively as I pestered them with deadlines and the details of getting each issue out.

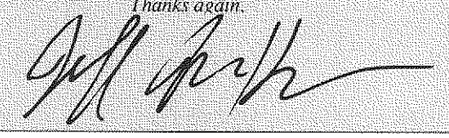
I also want to thank the members of the Editorial Board for their efforts, in particular in encouraging me as I struggled to get The Hotline off the ground. I especially thank ***Adele Goldberg*** *and* ***Dave Thomas*** *for listening and for their valuable suggestions.*

Another note of thanks is due to SIGS Publications. ***Rick Friedman*** *and his staff have supported my efforts without fail, as I learned the hard way what it takes to produce a monthly newsletter of this scope.*

However, the most important thanks goes to you, the readers. I have received considerable comment from many of you, and your encouragement made all of the effort worthwhile. I believe that The Hotline has met a need of yours, and I'm sure that under Marie's guidance it will continue to do so.

Good luck with your continuing exploration of the world of objects.

Thanks again.



What's in an object anyway?

Andrew J. Novobilski

MUCH HAS BEEN WRITTEN about object-oriented programming languages as an enabling technology for software reuse. An interesting subtopic for discussion concerns the tools available to aid in enabling class library reuse. This article examines one of the basic technologies available for working with class definitions: a class browser. It explains why a class browser is necessary, gives a brief example of a class browser, and presents the use of the browsing tool from the perspectives of the engineer new to object-oriented programming, the class consumer, and the class producer/maintainer.

The scenario

Software developers have traditionally been able to avail themselves of many different tools for examining procedurally-oriented code. Built to work with the basic grouping of source, the *file*, these tools would be used by the software engineer to answer such questions as *"Where is this function defined?"* and *"Where is this function used?"*

What makes an object-oriented programming language different is that an additional basic grouping of source code, the *class*, is introduced. Hybrid languages, such as Objective-C and C++, extend the C language to include an object model that supports encapsulation and inheritance via class definitions. Although conceptually straightforward, these extensions add several forms of complexity for software engineers when programming in the hybrid language.

First, the concept of inheritance is implemented to allow behavior from a single class to be located across several classes (which possibly means several source modules) according to a predefined hierarchy. Second, the behavior actually invoked during a message send is determined by a set of rules that allow methods to be overridden, with overridden methods being able to be accessed through a special message mechanism. Although simple in concept, this set of added complexity is very difficult for a programmer to work with unaided during a practical application development cycle.

The purpose of a browser

A browser, simply put, is a tool that enables the software engineer to deal with the added complexities of an object-oriented language. It allows a software engineer to explore the class hierarchy in its logical form without having to understand the exact implementation of the underlying language's object model.

A browser will provide three general types of information regarding a class hierarchy via its user interface. The first type of information allows the software engineer to explore the contents of a class hierarchy in a top-down fashion, starting with classes and moving toward the actual source that implements methods. The second type of information provides cross-referencing of the contents of a class hierarchy by usage. The third type of information provides the software engineer with analysis information regarding how the members of a class hierarchy interact with each other.

It's interesting to consider what the cost of this information would be in engineer time without the availability of a browsing tool. For example, consider what would be involved with locating the actual definition of a particular method available for use with an instance of a particular class. The engineer would start with the particular class in question and check its definition for the method of interest. If the definition of the method is not found, then the engineer proceeds to the classes' superclass and starts the search process again. Eventually, the location of the desired method will be discovered.

Although the above example is almost trivial, it is representative of the detailed indexing needed when working with an object-oriented language. By shifting the time-consuming (and error-prone) mechanical work of dealing with the implicit rules of the object model to the browser, the engineer is encouraged to spend time exploring the contents of and interrelationships between members of a class hierarchy.

An example

There are several class browsers currently available. One such tool, the Objective-C Browser, is an example of a class browser designed to work with a hybrid C-based language. Figure 1 shows the Browser's index panel with an example of each of the three above-mentioned types of information (top-down, cross-reference, and analysis).

For instance, the top-down information regarding the class hierarchy is available starting with a list of classes contained in the class hierarchy. From the class name the user is able to access information in greater detail by making selections from a set of predefined menus. Each selection narrows the scope of the information presented so the user is able to explore the hierarchy at the level most appropriate for the task at hand.

The cross-referencing capabilities of the Browser allow the software engineer to study the impact a proposed change on one member of the class hierarchy might have on other members of the same hierarchy. Cross-referencing information is provided by the Browser for items within a class hierarchy such as functions, methods, classes, and files contained. This allows the software engineer to identify possible problem areas within a class hierarchy when a modification to the source code is required.

The analysis capabilities of the Browser allow the software engineer to obtain information relative to the structure, layout, and interrelationship of members in a class hierarchy. This includes information that reflects the inheritance relationship between classes, information summaries concerning the contents and usage of selected items in the class hierarchy, and usage diagrams that reflect where message sends and function calls are resolved.

General purpose productivity gains tool

There are three areas of particular interest when discussing the role of a class hierarchy browser to enable productivity gains in the development of software using object-oriented programming techniques: the browser as an educational tool for software engineers new to the concepts of object-oriented programming; the browser as a tool for exploring already existing class hierarchies for possible inclusion in a new project; and the browser as a tool for maintaining existing class hierarchies during the maintenance phase of the software life cycle.

An educational tool

In general, the demand for engineering talent familiar with a new technology outstrips the supply of available engineers with the required skill set. One way to meet this demand may be to locate

Large companies supporting OOT

With endorsements from Lotus Development Corp. and AT&T Corp., the New Wave object-oriented environment from Hewlett-Packard Co. has moved one step closer to becoming a de facto standard. ... "Lotus' endorsement is a major, major addition," said Ruann Ernest, HP's director of marketing. ... While these new endorsements do not constitute the establishment of a standard, they move New Wave closer to the mark. ... The process is also reciprocal, since the closer something gets to being a standard, the more support it picks up; and the more support it gets, the more users and hardware makers will endorse it.

Source: "Lotus, AT&T Jump on New Wave bandwagon," COMPUTER RESELLER NEWS, *Philip H. Rose, March 19, 1990.*

Despite New Wave's potential, analysts are cautioning developers to temper their enthusiasm. ... "New Wave may be innovative and sexy, and a leading-edge example of where today's technologies are heading, but until Microsoft [licenses it] for Windows or Presentation Manager, it's not likely to become a 50 percent market share product," said Jonathan Yarmis, vice president of PC services at the Gartner Group Inc., a market research firm in Stamford, Conn.

Source: "Raft of Software Slated to Debut for HP's New Wave," PC WEEK, *Gina Smith, April 9, 1990.*

Microsoft Corp. is hammering out new application program techniques to help developers create more integrated "object-oriented" applications for Windows and OS/2 Presentation Manager (PM). ... With these proposals, Microsoft aims to deliver a portion of the object-oriented features of Hewlett-Packard Co.'s New Wave environment for its own Windows and PM environments...[this proposal] is part of Microsoft's long-term plan to create object-oriented versions of OS/2 and Windows.

Source: "Microsoft Seeks New Wave Power for Windows," PC WEEK, *Gina Smith, April 9, 1990.*

Ingres this week will announce a hybridized version of its 4GL application development environment that combines a Macintosh-like graphical user interface with object-oriented fourth-generation-language code.

Source: "Ingres to Unite Graphical Interface, Object-Oriented 4GL," DIGITAL REVIEW, *Steven Kovsky, April 23, 1990.*

Anti-hype

Just about everyone is jumping on the bandwagon, claiming they've got an object-oriented something or other in their software products. ... If you suspect there's an awful lot of hype in all this, you're right. Many of the claims made under the object-oriented banner are well out of proportion to what is really being delivered by the products. Still, with all that horse manure, there's got to be a pony in there someplace. ... Needless to say, the key to getting value from object-oriented software lies in isolating the benefits that the products can deliver rather than buying into the marketing hype that's used to sell them.

Source: PC MAGAZINE, *William F. Zachmann, April 24, 1990, pp. 83-84.*

Indeed, object-oriented has recently become the software industry's equivalent of oat bran. "Everybody and his brother is stamping object-oriented all over their products" whether it really is or not, said David A. Taylor, a vice president at the Servio Corporation, which makes object-oriented programs for storing information.

Source: "The Move to Modular Software," THE NEW YORK TIMES, *Andrew Pollack, April 23, 1990.*

[Proponents of object-oriented programming] claim that OOP systems will simplify developing complex applications; that changing and enhancing parts of an application can be done without "ripple effects" that can break other parts; that self-sufficient objects will make teamwork easier on large projects, without the need for constant communication; and that an object created for one application can be reused in another. ... The experience has been successful so far, but Lanham [Sarah, software engineer, Engineering Research Associates] was surprised to find that development took much longer that expected.

... The inheritance mechanism has proven fairly effective in making the system maintainable, but it's been used in a limited way. ... As these experiences show, OOP systems do provide at least some of the promised benefits. OOP languages may be easier to use, but they definitely take time and effort to learn. ...programmers and software designers have to learn a whole new way of thinking — and unlearn an old one — to take advantage of object-orientation, a process that can take months. Your productivity will drop drastically at first if you convert to OOP, rising again as your staff learns the new view of the world.

Source: "Objective Reality," UNIX WORLD, *Alan Winston, April 1990, pp. 72–74.*

The alliance that formed [between older and newer programmers] was enthusiastic and tense. The results have been languages like C++ and its brethren. All are much more procedural than their purer antecedents. They tend to mix enough object-orientation to make a difference, but none buy the whole package of beliefs. So that's why I say object-oriented programming is an instant lie. What has caught on these days is not exactly object-oriented. It is more "object-preoccupied." Or perhaps "object-tolerant." But you can't say that, can you?

Source: "Instant Lies," COMPUTER LANGUAGE, *P.J. Plauger, April 1990, pp. 17–20.*

Covia's chief information officer, Mark Teflian, elaborates: "Object-oriented programming will be constrained in growth by a lack of front-end development tools and the disparateness of existing operating systems."

Source: "25 Tough Integration Problems and Solutions! Object-Oriented Programming," SYSTEMS INTEGRATION, *April 1990.*

Object-Oriented Roundtable Opens: The Object-Oriented Programming RT, hosted by Mark Ericson, is open on GEnie, hosting discussions on C, Smalltalk, Actor, Pascal, and other object-oriented programming languages. Type <OOP> or <M165> for access.

Source: COMPUTER SHOPPER, *Michael A. Banks, April 1990.*

quality products quickly and craftsmen looking for ways to fabricate faster. Using terms Brad introduced, I can describe the two camps as follows:

The abstract camp:

- Craftsmen fabricating products one at a time.
- The quality of both the individual and the language is responsible for the quality of the product.
- Prototype (i.e., code) until you like it, then ship.

The engineering camp:

- Specialists assembling products from components.
- The process and use of appropriate, multiple tools are responsible for the quality of the product.
- Design, evaluate, prototype, and evaluate until you like it, then design the product and the test tools and implement the product.

In my opinion, progress from the abstract to the engineering approach is being hindered by consumers, or managers, of software development looking for a technical panacea, programming languages, CASE tools, etc., for a problem that is rooted the process and the organization.

The SCOOP-East conference was interesting for me in that the fraction of presentations advocating the engineering approach was higher than I have seen previously. Most of the tool-centered presentations, concerning languages and databases, seemed to be of the abstract camp, while presentations by Jeff McKenna, Nancy Martin, Ivar Jacobson, and Brad Cox were clearly in the engineering camp. I believe this trend in presentation content is consistent with Brad's model for the path of the Information Revolution. ■

Doug Bennett works at DesignWays, a company offering training and consulting services aimed at improving processes and organizations that develop software products. He can be reached at DesignWays, P.O. Box 581, New-town, CT 06470, (203) 425-9511.

Publicity Corner

Clippings summary

The number of clippings totalled 147: 57 articles in the monthlies, 16 in the bimonthlies, 41 in the weeklies, 29 in the biweeklies, and 4 in the dailies.

Approximately 33% were product announcements or reviews; approximately 27% were about O-O programming, languages, or tools; 20% were marketing news or hype; and the remainder was comprised of general technical articles, editorials, and articles on graphics.

Applications

It would be beneficial to have an object-oriented database such that when a construct in an object-oriented environment was created, a similar structure would be created in the underlying object-oriented database. This could be pragmatically accomplished by building this capability on top of a high-quality relational database. ... One would have the capability of a stable storage environment for inheritance taxonomies, methods and messages, in addition to what the traditional object-oriented environments provide in the virtual address space. Creating an object-oriented database transparently out of a relational database and integrating it with an object-oriented paradigm is practical with today's technologies. ...

Taken to its logical conclusion, the marriage of AI to relational databases provides an ideal foundation for generating an intelligent and high quality application environment.

Source: "Databases, Knowledge: A Union," **DIGITAL NEWS,** *Michael Stock, April 16, 1990.*

Object-oriented design will likely emerge over the next few years as the way to build interoperating applications in an open, multivendor, distributed computing environment, according to proponents of the technology speaking at the fourth annual Seybold Technology Forum [in Cambridge].

Source: "Forum Examines Object-Oriented Design," **UNIX TODAY,** *Mike Azzara, April 16, 1990.*

Object-oriented design principles combine with advanced database features to create Objectivity/DB, which can be considered as an extension of a workstation's operating system. The operating system is a suite of facilities, including memory management, process management, and file input/output control. Objectivity/DB extends this suite of facilities to include data storage and management at the object level, so that application programs can store and retrieve data as needed.

Source: "Engineering Database Operates on Object-Oriented Programming," **ELECTRONIC DESIGN,** *April 12, 1990, pp. 189-192.*

Newsworthy revelations

Les Hellenack, an analyst at International Data Corporation, projects that the market for object-oriented databases will rise from a tiny $6 million in 1989 to a respectable $250 million by 1994. ... That increase may not be enough to challenge the $2.5 billion relational database business. But for a few small firms and progressive users, object-oriented databases could be the next big thing.

Source: "Soft Talk: Objectivity," **COMPUTER SYSTEMS NEWS,** *Evan Schwartz, April 30, 1990.*

"You've got to think small. Too many people see the technology and like it, and they take this go-for-broke attitude. That's a catastrophe! You see, whatever implementation you choose right now, it's wrong five years from now. You've got to decide what behavior you want, and identify the fundamental underlying enabling technologies." [Adele Goldberg, speaking at the recent Seybold Technology Forum.]

Source: "Forum Examines Object-Oriented Design," **UNIX TODAY,** *Mike Azzara, April 16, 1990.*

engineers with backgrounds receptive to the new technology and educate them in its application. In the case of object-oriented programming, one difficult part of this education is teaching new engineers (used to working with procedurally based programming languages) a different set of fundamental concepts for software development.

In its role as an educational tool, the browser allows a new engineer to interactively explore the contents of a class hierarchy to gain insight into how object-oriented programming extensions affect the procedural language. For instance, the browser is capable of displaying not only methods defined in a class, but the methods that are inherited from a superclass. In essence, the engineer is taught to view the class, having a certain set of behaviors, as the basic unit of modularity regardless of where the behaviors are actually defined.

In additionally, the browser can be used to follow a stream of inquiries about the contents of a class hierarchy. The engineer can selectively view the contents of cross-referenced items such as methods, functions, and variables to gain an understanding of how they interrelate. This understanding can also be extended to include where messages sent to self and super are actually resolved.

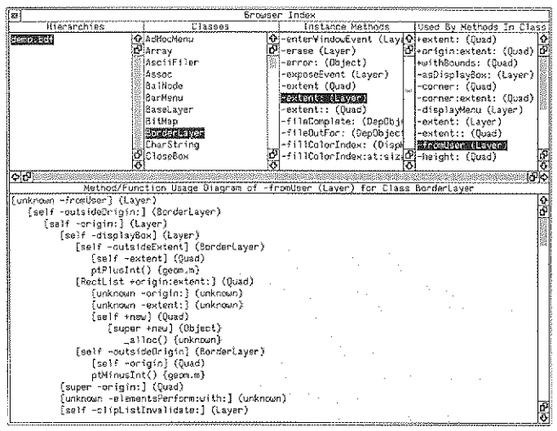


Figure 1. The Browser's index panel.

Class reuse tool

There are two types of relationships between a software engineer and a member of a class hierarchy: the producer and the consumer. The producer of a class is the one who defines, develops, tests, and documents a particular class for general use. The consumer of a class is the one who reuses a previously constructed class to aid in solving the problem with which he or she has been tasked. Significant gains in productivity are realized when the software engineer spends more time consumer-oriented (by constructing solutions using prebuilt and pretested software components whenever possible) than producer-oriented (custom recreation of each part of solution to a problem).

A browser serves as a tool for encouraging consumer orientation by providing a software engineer with the ability to browse what's available in a preexisting class hierarchy to see if it might meet a required need. For instance, an engineer tasked with developing an application that allows manipulation of a list of items can use the browser to check a preexisting user interface class hierarchy for components that can be used to construct the required interface. After browsing the list of available classes, most likely candidate classes can be explored in greater detail. If there isn't a single class that completely covers the requirements, then the engineer can browse the contents to see if there are a series of components in the class hierarchy that together would solve the problem. The browser is then used to check for interrelations between class (through cross-referencing) and what is required to make the classes work together (through analysis).

Class production and maintenance tool

One of the requirements to transform the ideal of software constructed from reuseable software into a widespread reality, is the availability of said reusable components. The class producer is tasked with the job of developing and maintaining classes that will have general usage in the software development community at large. The browser is capable of supporting this task in two basic ways: as a tool for ensuring new components fit with existing components; and as an aid in the task of software maintenance.

One method of developing new software components is to search a current hierarchy for classes that almost fulfill the stated requirements. If one is found, it would then be possible to extend these classes through inheritance to incorporate the desired behavior. The browser can be used to verify that the extensions obey class protocols, and that new methods are named to preserve the polymorphic attributes of the existing class hierarchy.

For instance, building a jet airplane class from a preexisting airplane class would require a careful analysis of what is available already as an airplane, and what additions or modifications would be necessary to build the jet airplane. The browser would allow for the exploration of the current definition of an airplane to find out how the class works by answering such questions as: *"Which methods does the superclass depend on the subclass to support?;"* and *"If we want the instance of a class to have a particular behavior, what is a good choice for a method name that will ensure consistency with other classes?"*

The second major use of a browsing tool for the class producer is in the maintenance of existing class hierarchies. The cross-referencing capabilities of the browser quickly highlight where a change in one area of the hierarchy will affect other areas. In addition, analysis capabilities such as method/function traces provide a starting point when searching for anomalies in the behavior of a particular component.

Part of the bigger picture

It is important to mention that, although a powerful tool, a class browser is only part of the overall object-oriented programming language development environment. The strengths of a class browser — coupled with the availability of a reuseable class library manager, a debugger, an interpreter, an inspector, a performance analyzer, and other tools — will support the effort to enable software component reuse at a much larger level. It will be the enabling of this reuse that will help realize the tremendous productivity gains that object-oriented technology promises.

Andrew J. Novobilski is a senior member of technical staff at the Stepstone Corp. Author of the Objective-C Browser, he is currently involved with the design and development of tools for enabling software component reuse. He can be reached at The Stepstone Corp., 75 Glen Road, Sandy Hook, CT, 06482; (203)426-1875.

the problems, we feel that a good compiler should be able to eliminate a large portion of the difference in *size* and *speed* between an Eiffel program and an equivalent program coded in C. This would involve a good many separate actions and improvements, but each would be reasonable and the net result should be sufficient.

Any new implementation of the compiler would have to have three characteristics:

1. It would have to provide for separate compilation of classes so that developers could assume a reasonable and predictable cost for making changes. This would imply a new OOP consistency check step between compilation and binding.
2. The compiler would have to permit targeting of "object" files so that separate systems could be kept from colliding.
3. It would simply have to be faster by being smarter.

The tooling in the Eiffel environment — at least at the time we used it — was totally inadequate. It is hard not to turn to other O-O development products as models of what the tooling in support of "programming in the large" should be in the OOPS world. We certainly consider a browser to be the lower bound of acceptability. In terms of other "standard" tools — source control, configuration management, and so on — we would at least be a great deal happier were Eiffel more cordial in its relations with these standard tools. A "less intelligent" compiler of the sort described above would go a long way in making Eiffel a tool-friendly language.

Steps to success

It is clear from the Cognos experience that it is quite as possible to do it wrong in an OOPL as in any other language. There has been considerable introspection at Cognos on what went wrong, why it went wrong, and how it might not have gone wrong. It is tempting to blame all our problems on Eiffel. This would be unfair and untrue. At least half the problems were of our own making. Rather than indulge in an orgy of breast-beating, it is more useful to frame the experience in terms of advice to those considering the use of an OOPL.

Our experience with Eiffel...

Positives:

It is important to state at the outset that our experience with Eiffel was not an unalloyed failure. Eiffel embodies the virtues claimed for OOPL's, and the reasons for which it was chosen were supported in practice.

Correctness. Eiffel provides many mechanisms — preconditions, postconditions, invariants, variants — that a programmer may use to write code assured to be of higher quality than would be obtained with equivalent effort in traditional languages. Further, the thorough checking by the compiler catches many potential bugs. Many programmers voiced the view that when using Eiffel, only the serious bugs — the fundamental design errors — made it past the compiler. Since no compiler is yet psychic, that is as much as can be asked.

Portability. Because the Eiffel compiler controls the nature of the intermediate C produced, handles pointers, and takes care of many of the other items which can lead to nonportable code, the programs written with Eiffel tend to be portable. It is still possible to make Eiffel-based code nonportable; but some effort is required, and it is relatively simple to identify potential problem areas.

Mutability. Programs change, and the maintenance or enhancement of existing code is always a difficult business. We had several instances in which large portions of the application we were developing were changed in radical and significant ways. It was possible to do this with confidence because encapsulation works.

Productivity. Once our programmers had made the paradigm shift to OOP they were more productive — sometimes too productive. Further, we know that we did not have a chance to realize the full productivity potential of OOP because we didn't get to exercise the value of reusability on a second product. We did not have a source-level debugger until late in the project, but our limited experience showed it to be a tremendous productivity multiplier.

Negatives:

Our biggest single problem with Eiffel was with the performance of both the compiler and the programs the compiler produced. We also had problems arising from the way in which the compiler implemented the checking of system consistency required to achieve valid runtime binding.

Compiler performance. In terms of the compiler, we know that for a single class the compiler performance could probably be doubled with a better implementation. This is not the only problem. The compiler processes individual systems in the context of a system. We conducted various experiments that suggested that the checking of class consistency *at compile time* can account for a third or more of the time to compile a class.

Program performance. The programs we produced with Eiffel were essentially unusable. They were far too large and, partly as a consequence of the size, ran far too slowly. Again, our analysis of the programs suggests that both size and execution time could be substantially reduced with a better compiler implementation. ISE has made some progress in this regard, but less than we would consider sufficient.

Variant handling. We had ample reason for having more than one instance of many classes. For example, we were writing a system targeted to many hardware/operating system environments. The practical handling of this variation required that there be more than one realization of many classes. In a variety of ways, the Eiffel compiler made the handling of this variation — both at the level of source control and of configuration management — excruciatingly painful.

Class reuse. One of the touted values of OOP is that classes can be reused. We found this true for Eiffel — but only if we copied the class source. Again, the compiler made assumptions about the way in which software systems are constructed that made safe and proper reuse impossible.

account for why they all have the best product, education, and service? Does this imply that, despite all the techno-babble, they really have the same or very similar products?

Cruising the exhibit area gave me a chance to try and unravel this mystery, but with limited success. When the questions met the road, I was sometimes told the answers to my most critical questions were proprietary. On the other hand, some of the demonstrations were very impressive and the vendors very knowledgeable. While presentations can be informative, there is no substitute for seeing a fully functional product in action. More than adequate exhibit hours made it possible to do so in an unharried manner.

On a more serious note, I do have one major reservation concerning SCOOP-East. The food provided at the lunches wasn't what one would expect given the registration fee. In anticipation of what an English version of SCOOP cuisine would be, I am planning to take a case of Twinkies to SCOOP-Europe.

Despite mediocre lunches, I believe organizations represented by the attendees received tremendous value for the expense and time involved. In a demand-driven technology such as OOP, there is a great temptation for vendors to tell you what you might like to hear: namely, their technology will solve all of your problems, enabling you to fulfill your mission with a fraction of the effort currently required. Given the relative youth of the industry, it is often difficult to evaluate these claims. In such an environment, there is no substitute for developing in-house expertise, and SCOOP-East was an excellent place in which to do so.

Dr. Jacob Stein is Chief Technologist at Servio Corporation, where he has been active in the research, development, and commercialization of object technology for six years.

Brad Cox lecture at SCOOP-East

Doug Bennett

Brad Cox, of Stepstone Corp. and Objective-C fame, gave the invited lecture on the last day of the SCOOP-East conference at the Wang Institute of Boston University. His talk was well attended and, on average, politely received. In this writer's opinion, the thrust and content of his remarks accurately described the direction of change in the industry and correctly reflected the tension evident in the SCOOP conference itself.

That tension was demonstrated in the comments of two attendees: the person behind me said the lecture was "the best part of the conference." On the other side, the first question asked at the end of Brad's remarks was something to the effect of, "Isn't compile-time type checking the thing that will guarantee that the shuttle will safely land on the moon?"

The content of the lecture fell into three sections: the nature of software; an analogy between the Industrial Revolution and the rise of mass production (use of "reusable" parts) and the Information Revolution that we now have the privilege of experiencing; and the role test and specification tools will play in the Information Revolution.

Fred Brooks said that the essence of software is that it is complex, nonconforming, and changeable. Brad suggested that those terms are not the essence of software, but are the results of how we handle software. He described the tension between those who view software as abstract, like mathematics, and those who believe that software is tangible, like other engineered products. Brad's position is that the essence of software lies between the tow extremes: a swamp that is "too thick to drink and too thin to plow."

He introduced the idea that levels of software components be analogous to electronic hardware: gate, block, chip, board, and rack level components have direct analogies in software. The objects in programming languages correspond to the chip level components in the hardware world. While he did not explicitly say so, the clear implication is that a product useful to end users must consist of several levels of objects, both above and below the objects in a particular programming language.

The adjective "object-oriented" has a different meaning within the context of each level: just like the adjectives "large" and "small." His definition of "object-oriented" that was valid at any level was "anything that enables people to reason about the problem using common-sense techniques."

Brad used the gun manufacturing industry as the basis for his historical comparison. The key features of the development of interchangeable parts in the gun industry that are relevant to our current situation include the following.

Up to the last part of the 18th century, guns were made "from scratch" by craftsmen. "Scratch" meant forging a rod from a billet so they could file it into wood screws. The change to assembly from standard parts was motivated by the purchasers of guns, not the gunsmiths.

The technical solution, making lots of "standard" parts, failed for 20 years until the advent of specification and test tools, i.e., gages, that ensured that the parts were, in fact, within acceptable tolerances. The time scale for making the change from fabricated to assembled product was about 50 years.

Brad's point was that software is still being fabricated by craftsmen filing screws from solid steel (programmers writing lines of code). Some of the tools already exist to build software components, but the main obstacle to change is cultural. He concluded by saying that when there are large economic forces dictating change, e.g., the crisis in software productivity and quality, change is inevitable.

The entire SCOOP-East conference was permeated by the tension between consumers of software products looking for ways to assemble

cation of techniques for finding information is dictated by the representation scheme. Hence, the extent to which object-oriented software engineering, as compared with traditional software engineering, is more or less able to support general retrieval of reusable artifacts will be dictated by the (as yet to be devised) representation scheme.

One of the biggest inhibitors to successful reuse is the inability to understand (and trust) the work of others. Tools are needed that permit the user to probe the retrieved artifact by inspecting it in a variety of ways, asking questions about it, cross-referencing its components, and finding examples of its use. Such tools are typically delivered as part of a well-integrated object-oriented development system.

Conclusion

As we have seen, there are many factors that influence the successful practice of reuse within an organization. Next month, we review a number of case studies of reuse within organizations. In preparing the studies, we asked a number of managers and developers the following questions:

1. *Operational definition of reuse*

   How do you define reuse?

   Do you agree with the definition given in the July column?

2. *Forms of reuse*

   What kinds of components do you currently reuse or expect to reuse?

3. *Experience with reuse*

   Do you have a special library of reusable artifacts?

   Is it the file system or a database or something else?

   Who takes responsibility for reuse in your organization: managers, developers, or someone else?

   Is there a corporate librarian for your reusable components?

   Where do reusable assets come from?
   - purchase commercial products from outside organization?
   - purchase from outside as part of development contract?
   - obtain from another project within organization?
   - create within development project?

   What are requirements, if any, for documentation of reusable components?

4. *Managerial issues*

   What are the managerial concerns about reuse in your organization?

   Do you have a special reward structure to encourage reuse? to encourage the creation of reusable assets?

   How does an emphasis on reuse change the performance review and reward structures for individual engineers?

5. *Developer issues*

   How do you inform team members of what reusable assets are available?

   What tools do you have for locating and understanding the reusable software you use?

6. Do you believe that reuse is difficult to achieve and, if so, why?

By the time you read this article, our initial case studies will have been completed. But we are always interested in adding to our knowledge base and, where possible, sharing with other readers. If you wish, please answer these questions and write to us! ■■

Bibliography

1. Biggerstaff, T. and C. Richter, "Re-usability Framework, Assessment, and Directions," *Tutorial: Software Reuse: Emerging Technology*, IEEE Computer Society, EH0278-2, p. 3–11.

Dr. Adele Goldberg is President and CEO of ParcPlace Systems. Adele spent 14 years as a researcher and manager at Xerox Palo Alto Research Center. She has served as president of the ACM Systems Software Award.

Kenny Rubin is a computer scientist at ParcPlace Systems, where he has been a primary curriculum developer of three of the ParcPlace Systems Smalltalk-80 training courses; he has written on object-oriented technology, artificial intelligence, and human/computer interaction.

Both authors may be reached at ParcPlace Systems, Inc., 1550 Plymouth Street, Mountain View, CA 94043; (415) 691-6700.

SCOOP-East *continued from page 1*

To my great surprise, SCOOP-East did not meet my expectations. It was much better than I'd expected. By and large, there was very little marketing hype in the tutorial program, and the presenters were present for the entire conference. There really was opportunity to catch them in the hallways and at receptions to ask questions you didn't feel comfortable asking during their tutorials. Perhaps the venue, the Wang Institute of Boston University, being located outside any major city center, contributed to the casual conversations between attendees and lecturers.

Comparing SCOOP-East to OOPSLA, I found the atmosphere at SCOOP-East much more conducive to actually learning about the technology. OOPSLA is a great gathering for those involved in research and development of O-O systems; unfortunately, its size (2,000 attendees in New Orleans) and frenetic pace make it less optimal for those seeking to increase their knowledge about applying the technology.

The tutorials covered a very broad spectrum of O-O topics from the very technical to managerial. Among the topics covered were: programming, analysis, languages, management issues, integration of the technology into organizations, design, data management, and commercial applications. Both neophytes and sophisticates could find a wide selection of useful tutorials. I suspect that many of the instructors benefited, or could have benefited, from attending the tutorials. I also suspect that many first-time attendees will find attending a second SCOOP to be as informative as attending their first.

As I mentioned earlier, the important thing about the vendor presentations was that they were easily discernable from the tutorial program. Often, they were very informative, even humorous. When several vendors each have the best financing, people, product, education, service, and the like, one starts wondering: Do they all have the same source of financing? Are the same "best" people working for all the vendors? Does this

Two things should be noted about this advice: First, it sounds trite. That is often the case with essential wisdom. When it is stated, it seems obvious — in retrospect it seems painfully obvious — but it was not obvious to us when it should have been. Second, most of it has been said before — some of it here in THE HOTLINE.

Walk before you run — don't be too ambitious.

Although the initial project team and many of the early recruits had some prior experience in OOP, there had never before been any product-oriented OOP development. This was no deterrent to leaping into the development of as large and complex a product as any the company had ever previously undertaken, in a language in which nobody had more than a few weeks' experience. Not smart!

Learn through prototyping — build something to discard.

As part of the review of our involvement with OOP, we looked into the possibility of employing some of the more modular elements of the system we had tried to produce within our current product set. The one that would have been most useful was not usable, largely because it would have required redesign and rewriting to exploit fully the benefits of object-oriented technology. The people who produced it had leapt into full-scale development before they had made the paradigm shift to object-orientation. Had they had a chance to learn the craft by doing a throw-away prototype first, they might well have spent the same amount of effort and have ended up with a usable component instead of an inspiring prototype.

Define through prototyping — surface reality is user's reality.

One of the major drawbacks of the waterfall methodology used with traditional languages is that the clients of the system do not have a chance to deal with it, except as a set of abstract written specifications, until it is too late to make any significant changes. This leads, all too often, to products that meet the spec but not the need. It is much easier with OOPLs to create prototypes with "smoke and mirrors" functionality that allow users to test drive the proposed products — albeit in highly constrained ways. This is invaluable for determining what is desired and provides designers and developers with highly realistic criteria for the character of the end product. We failed to use this technique on our OOP-based product; ironically, another product developed at the same time without OOP did use extensive prototyping and has been both a technical and commercial success.

Pseudocode is real code — create language hierarchies.

It may well be asked whether a new class is a component of an application, an addition to a library, or an extension of the language. I would argue that every new class is — and should be thought of as — a language extension. Whenever I define a set of classes oriented to a specific problem domain, I have created a problem-specific dialect of my base language. This perspective on OOPLs has an important implication for how they may be used to address real problems, i.e., how we create programs to meet needs.

Most people I know tend to write their program descriptions in a "pseudo-code" that is then laboriously translated into the implementation language. Others have expended great effort in defining "specification languages," in which problems are formally stated — with the expectation that the formal specification can then be more or less automatically translated into the *real* language of the system. This is the basis of much of the current excitement with CASE tools. The thing that OOP does is allow us to write our problem specifications in terms of problem-specific classes that form a problem domain language. The challenge of implementing the domain language consists of defining another domain language, closer to the target OOPL, that realizes the original domain language. By an iterative process, the hierarchy of domain languages can "reach down" to an existing OOPL. This sort of approach makes the distinction between pseudo-code and real code vacuous.

To put it in another way, if I am asked to write a payroll program, I should not write a payroll program. Rather, I should design a language tailor-made to write payroll programs. Once I have implemented my payroll-writing language, writing the payroll program is trivial. Not only that, *changing* the program — which I am certain I to do — is equally trivial.

It would be possible to argue forever about whether design should start with the problem and work to the language or vice versa. It all has too much of a sense of *déjà vu*. The point is that the intrinsic extensibility of OOPLs allows us to think of all programming as defining languages that naturally address problem domains.

Design anthropomorphically — use people/actors, not processes/activities.

OOP is frequently described as being anthropomorphic in nature, and there is a very natural mapping of the object-oriented idiom onto problems with which business systems deal. There is a

> ***For OOP to deliver on its promise, the technology itself requires strengthening. Beyond this, there must be an effort on the part of management to understand the technology and to manage its adoption and use.***

problem. The MIS industry has spent the last 30 or more years training its clients in procedural thinking so that users will think about their problems and activities in the same way that systems deal with them. We have successfully indoctrinated an entire generation of clients into thinking about working with computer systems in an unnatural way. If object-oriented applications are to be employed, we must undo the damage. The desktop interface developed by Xerox PARC and popularized by the Macintosh is a great start. With its graphical interface and nonsequential control structures, this model is allowing users to revert to a more natural way of thinking about and working on problems.

As practitioners, we must go further. Systems designs are best if they abstract the functional characteristics of existing actors into isomorphic computational entities. The OOP class is a natural way of doing this. The simulation languages to which OOP owes so much had no problems with such anthropomorphic design. Given that business systems can be considered as full function simulations driven by a "live trace," the anthropomorphic model is eminently desirable.

Less is more — don't build what you don't need.

It is frequently asserted that OOP leads to greater programmer productivity. The experience at Cognos supports this assertion. There is, however, a negative aspect to this productivity against which it is necessary to guard.

The first manifestation of the dark side of OOP productivity is what might best be termed "cancerous classes." Adding a new feature (method) to an existing class is almost trivially easy. Consequently, there is a temptation for developers to add many "just in case" features. Each in itself is benign, but the aggregation of all the benign features is definitely malignant. The only way to avoid this phenomenon is to build only what is needed. To *build* only what is needed it is necessary to *know* what is needed. To know what you need, you must design.

The second form of harmful productivity arises both from the ease with which classes and features may be created and also from the inadequacy of most browsers in helping developers to find classes that could be used or reused.

When we started with Eiffel at Cognos, we had no browser. When we stopped, we had an inadequate home-grown browser barely usable as a tool for discovering classes that implemented desired capabilities. Given the difficulty of finding reusable classes and the ease of creating new classes, we suffered an explosion of similar classes in separate parts of the system. Better design would have helped. Better tools are essential.

Activity ≠ progress; know when real work is being done.

As suggested in the previous subsection, it is possible to have a great deal of development activity that does not provide tangible progress. Yes, code is being generated; the class count is rising; but real progress toward the creation of a product is far below what the activity level indicates.

The measurement of progress is a difficult matter. It is often difficult to determine the difference between needless polishing and valuable refinement. A good example of this is in the inference of reusable classes. As has been pointed out elsewhere, it is often only during the development activity itself that opportunities for generalization of classes become evident.

In this circumstance, it is a good activity to take a set of classes and to restructure them to find a better inheritance tree or to eliminate classes through better packaging of functions.

Valuable as these activities may be, they are digressions from the main thrust of a project and must be managed explicitly. Anarchic refinement and polishing can drain the resources of a project to the point where progress ceases.

Abdication is failure — novelty is no excuse for not managing.

In a recent issue of THE HOTLINE (1:3, January 1990), Marie Lenzi argued persuasively for the need to scope and plan projects *even in the absence of adequate experience data*. It is entirely too easy for management to abdicate its responsibilities on the grounds that the technology is new and that it is therefore impossible to do the kind of detailed planning and milestone setting that is the norm for projects executed with traditional languages.

Yes, the schedules created for an OOPL-based project may be less credible simply because people are less skilled at scoping the work to be done and the effort required. It will take time to develop the explicit data and the implicit skills required to do sound planning and scheduling for OOP. Our management did a less-than-adequate job in this regard, and the cost was extremely high. Matters improved as the project progressed, but even at the end, goals were too vague to permit the necessary evaluation of whether the project was sound.

And in conclusion ...

The promise of OOP is great. It can and should be as much of a revolution in the area of software engineering as was structured programming. However, it is no panacea. For OOP to deliver on its promise, a number of things must happen: The technology itself requires strengthening — our experience with Eiffel showed this. Beyond this, there must be an effort on the part of management to understand the technology and to manage its adoption and use. Again, the Cognos experience, especially in contrast to the experience of some others, shows how vital this is. ■

Dr. Burton L. Leathers was trained in mathematical sociology and entered the computing field through work in statistical computing. He is now in the research department at Cognos working on object-oriented technology and distributed computing. Dr. Leathers can be reached at (613)738-1440.

strategy by leveraging the development efforts of previous corporate projects. Also, a corporation often adopts standards for software formatting and documentation. Projects carried out within the corporation will follow these guidelines, encouraging understandability through their consistent use. Outside vendors are less likely to follow these guidelines.

A word of caution. Vendor-purchased reusable libraries have been formally tested and officially released as "ready for reuse." It is all too tempting for one project to raid the repository of another project in the name of corporate reuse. All to often, code that was not intended or ready for reuse is propagated throughout an organization. Our June column in THE HOTLINE dealt further with this need to control access to the corporate repository of reusable code.

Many software development projects are carried out under contract. The desire to reuse and to obtain reusable artifacts poses a new set of managerial concerns. First, if reuse is expected, and the cost of the development is commensurate with reuse, then this expectation must be stated in the request for contract bids. If the corporation has acquired a library of reusable artifacts, and its use is required, then this fact must be included as well.

Such a desire, however, poses a potential problem for the contract developer. Many developers today see the opportunity to create software under contract as an opportunity to build their own reusable libraries. With the aid of a familiar library, the developer can bid more competitively because the development time is reduced. Who owns the reusable artifacts created as part of a contract? The answer has to be written into the contract.

Developer issues

There are two main social issues with regard to software developers. First is convincing them that reuse is a worthwhile endeavor. This is most effectively accomplished by changing the review and reward structure. Next, developers must overcome the "Not Invented Here" mindset. Specifically, developers oftentimes resist using something that they did not develop. With proper motivation, most developers can be persuaded to reuse.

Once developers have agreed that reuse is a worthwhile endeavor, there are three major engineering activities to address:

First, reusable artifacts must be intentionally designed and developed.

Second, reusable artifacts must be described, classified, and entered into (and removed from) appropriate repositories. Implementation artifacts must be maintained over their useful lifetimes.

Third, tools and processes must be developed that support finding, understanding, modifying, and composing artifacts. In general, these problems exist for developers using both traditional and object-oriented technologies.

Designing reusable artifacts

Designing reusable artifacts is technically challenging, with many tradeoffs to consider. Two factors control reusability of an artifact: choice of the abstraction level of an artifact, and complexity of the artifact. The more specific the artifact, the less likely it is to be generally reusable. The design problem is the degree to which the visible attributes of an artifact reflect a specific application domain. Generalized artifacts (ones that make few or no assumptions about the domain) tend to be more reusable at the expense of time or space or both.

Complexity of an artifact is measured in terms of the number of aspects that must be matched for reuse to be possible. An example of such complexity is the number of messages in the interface specification that must be supported, or the number of details that a subclass has to implement to fulfill the obligations defined in the abstract description of a framework.

Artifacts with many aspects are more difficult to reuse than artifacts with fewer aspects. To some extent, the design of a reusable artifact requires applying 20/20 hindsight to past project requirements and 20/20 foresight to future project requirements. As such, the librarian for the repository of reusable artifacts plays a central role in the procurement of these assets, choosing which are entered into the stable corporate-wide repository and which are still to be evaluated.

The librarian is a central resource of knowledge about the past and of expectations about the future.

Representing reusable artifacts

One of the most difficult problems with reuse is developing a suitable representation for artifacts. In particular, we would like a representation that encodes the semantics of artifacts in such a way that a user who is trying to solve a problem with his own knowledge and semantics can locate an appropriate reusable artifact. Such artifacts should be retrievable by multiple pathways to support the variety of ways in which users may access them. Furthermore, the representation should allow for a variety of different perspectives on stored artifacts, and should permit versioning and configuration/ management activities. Lastly, a representation should allow for partial and uncertain information. This allows artifact developers to evolve their designs over time by permitting well-defined aspects to be expressed with certainty, and less well-defined aspects to be left fuzzy.

Object-oriented software engineering has potential for making the representation problem tractable. This is due in part to object encapsulation and polymorphic protocols. Object encapsulation simplifies the semantics of reasoning about what an object does by reducing the scope of the problem to that of reasoning about the object's interface specification without having to reason about its implementation. Polymorphic protocols provide a criterion with which a classification scheme can index similar objects. Objects with similar protocols can be substituted if other semantic criteria are met. Hence, the classification scheme can use protocols to help identify similar objects.

The representation of reusable artifacts is currently a research topic.

Tools to support reuse

To be reused, artifacts must be found and understood. To find artifacts, developers describe what they want and tools decide how to satisfy the request. This simple declarative model is rarely achieved in practice. The simple reason is that most representations are insufficient to support sophisticated queries and reasoning. In general, the sophisti-

framework can be reused across applications. The Smalltalk-80 system includes a user interface based on a uniform model of "viewing" and "interacting" with objects. This user interface is called MVC or the model-view-controller framework. It is an example of a framework for implementing interactive front ends to software systems.

When a framework is reused, it often constitutes a substantial part of the application. The application obeys the protocols of the abstract classes, but specializes the implementation to provide concrete details. In the case of the Smalltalk-80 MVC framework, the concrete details include the algorithm for displaying a visualization of some aspect of the model object, and the algorithm for responding to user-initiated events (button clicks, keystrokes, movements of the pointing device).

Interactive graphical applications are created using the MVC framework by implementing subclasses that dictate how a model looks on the screen and how each button click or key press will be interpreted. The framework handles general functions, such as the overall interactive control, selection of parts of the visual display, and display update.

Another example is the Smalltalk-80 browser, whose role is to provide a viewing and interaction mechanism for retrieving, organizing, and editing source code. Judicious design of the browser application makes it an important reusable asset. The browser supports search functions, explanation ability, and interactive compilation of individual methods.

These facilities are implemented by sending appropriate messages to the objects that represent classes, source code, compiled code, and so on. By specifying the implementation of the browser such that it relies only on an abstract notion of class compilation, a class can specify an alternative compiler, different search and query functions, and unique explanations, allowing the browser to be partially language-independent.

Fully completed applications are also reusable artifacts. Text editors, network communications packages, file managers, picture editors, database managers, and so on can be reused as is, embedding them within other applications. However, when developed with object-oriented technology, these applications are implemented as collections of interacting objects, where each object is an instance of a class. The classes can be refined by creating subclasses. As such, completed applications are available for further refinement and are thus a form of framework.

An interface specification is the most abstract of the reusable components of a software system. It consists of a set of messages that embody a coordinated set of behaviors. Classes whose instances perform the role implied by these behaviors must provide an implementation for the behaviors. For example, the interface consisting of + - * and / represents an interface for objects that carry out basic arithmetic operations.

To add a new class such as Matrix that reuses the arithmetic specification, it is necessary to implement equivalent matrix operations for + - * and /. Another example in the Smalltalk-80 system is the object streaming protocol. The idea of a stream is one object that moves (or streams) over the contents of another object, keeping track of its current access position. To adhere to the streaming specification, a class must implement messages content, next, nextPut:, and atEnd. An example stream is an ExternalReadStream, which reads (next) and writes (nextPut:) objects to and from a file. A very different kind of stream is Random, which obtains the next number in a stream of randomly generated numbers.

Obtaining the benefits of reuse

An overall corporate reuse strategy is founded on two interrelated processes: the design and development of reusable components, and the utilization of reusable components. To promote these processes, both managerial and technical support must exist, as well as mutual cooperation. In particular, managers need to adjust the manner in which they supervise, review, and compensate software developers.

Developers must learn both to overcome personal biases against reusing artifacts that they themselves did not develop, and to understand what is involved in the development and utilization of reusable components. Let's address each of these roles in turn.

Managerial issues

There is currently no rigorous economic model that exposes the real benefits and costs of software reuse, nor is there a method for analyzing their situation. Current management of reuse has been based on retrofitting well-understood managerial practices used for other situations, and on experience gained through trial and error.

In addition to the economic reuse models, managers need to adopt a personnel review and reward structure that encourages developers to practice reuse. Without the proper motivation, developers have less desire to reuse. For example, a developer who is evaluated and compensated based on the number of lines of code written has little motivation to reuse someone else's code. A more useful metric is to evaluate developers based on timely delivery of the completed project, and on the amount of available reusable code they employ.

In addition, for some members of the team, evaluation can include the amount of new reusable code they generate. One can even imagine a royalty-based system, in which developers who generate reusable components are compensated each time their components are reused. (Certainly, vendors selling reusable class libraries expect to be rewarded on this basis!) Such systems provide the incentive for developers to make reuse a fundamental aspect of their jobs.

Managers must understand the various strategies for acquiring reusable software. There are several options available. First, a manager may consider purchasing a package from a third-party vendor. This has the advantage of avoiding the internal startup cost. Such packages should be purchased in source-code form, since it is rarely possible to purchase exactly what you need — refinement will be necessary and understanding (i.e., trust) is facilitated by having the ability to read the code.

Although purchasing the components does little to build inhouse expertise for developing reusable components, it does provide early examples of how to design for reusability. It also provides earlier opportunities for successful experience with reuse.

Another alternative is to obtain the reusable components from another project within the organization. This epitomizes the goal of the corporate reuse

Languages & Environments

The impact of Windows 3.0 on the OOP market

Zack Urlocker

On MAY 22, 1990, Microsoft announced Windows 3.0, the long-anticipated update to its MS-DOS graphical user interface. Windows 3.0's influence on the PC market is profound. For many corporate MIS departments, vertical market developers, and commercial "off-the-shelf" software developers, Windows 3.0 may be the force that brings object-oriented programming to the forefront.

A bit of history

Microsoft Windows was first announced in 1983 with the support of dozens of hardware vendors. The initial version was greatly delayed and topped the industry "vaporware" list. When Microsoft finally shipped Windows 1.0 in November 1985, the product was exciting, but not exactly a breakthrough. Like the early Macintosh, Windows 1.0 was underpowered and terribly slow on what was then a standard hardware configuration: 512K, dual floppy disk drive, CGA resolution graphics.

Developing a Windows program was difficult. The only development system available for Windows was the Microsoft Software Development Kit (SDK), which could be used with either the Microsoft C 4.0 compiler or MASM, Microsoft's assembler. There were no high-level languages available, no commercial libraries, no source code debugging, and little documentation.

Clearly, Windows development was not for the faint of heart. Even if you were willing to live with the memory constraints of Windows 1.0 and the lack of tools, you still had to write enormous quantities of code to perform the simplest of tasks. The classic "hello world" program took over 100 lines of code and had to be edited, compiled, and linked outside of Windows.

Object-oriented tools for Windows

In late 1986, The Whitewater Group, a startup company located in the Technology Innovation Center of Northwestern University, announced the second commercial application for Windows: Actor. Actor became available in February 1987 and was the first object-oriented development system for Microsoft Windows.

The Whitewater Group recognized that Windows would become an industry standard, edging out, among other competing windowing systems, Digital Research's GEM and IBM's TopView. The Whitewater Group also realized that Windows development was just too hard with standard procedural languages like C.

Early versions of Actor and Windows were limited severely by the amount of available memory on DOS. The fact that Actor 1.0 even ran on a 640K machine running Windows 1.0 was somewhat of a miracle. Actor and Windows evolved in subsequent releases and were able to take advantage of expanded memory. When Actor 1.2 became available on Windows 2.1, there was enough memory to create decent-sized applications.

Growth in the Windows market

There are both technical and business reasons why Windows 3.0 will change the PC market. Technically, the product is fantastic. The graphical user interface is far superior to previous DOS windowing systems and gives the Macintosh a real run for the money. In addition, Windows 3.0 breaks through many of the previous limitations of DOS. It provides virtual memory and breaks through the 640K barrier, providing much-needed relief to RAM-crammed network users. Windows 3.0 even provides cut&paste and limited multitasking of standard DOS applications. As Bill Gates put it, this makes Windows "a better DOS than DOS."

The significance of these technical improvements is that they enable and encourage users to run Windows as a permanent DOS shell. With previous versions of Windows, a user was likely to use Windows for a single application, such as Microsoft Excel or Aldus Pagemaker, and then exit Windows to run other DOS programs. Now with Windows 3.0, these same users are likely to use the Windows 3.0 Program Manager or File Manager as their main DOS shell and run both Windows and plain DOS applications from within Windows.

In addition, Microsoft has lined up the support of all major software publishers. All of the application categories, including spreadsheets, word processors, communications, database managers, and development tools, are available under Windows 3.0.

Even long-time Windows bashers such as Lotus and WordPerfect have committed to creating Windows versions of their software. Many software development firms that previously rejected Windows as a delivery platform are realizing that the market is moving towards Windows.

At the same time, Windows 3.0 is being bundled by dozens of hardware manufacturers. Zenith's commitment is especially noteworthy since it is installing Windows 3.0 on all hard-disk based PCs. As a result, there will be an estimated 10 million Windows users by the end of 1990, with further growth in 1991 and 1992. That's a booming market even in the software industry.

All of the growth in Windows is good news for end users. It means that

applications will become easier to use and provide better integration. However, for many MIS departments and vertical market developers, Windows 3.0 may seem a little disconcerting.

Upping the ante for MIS developers

The only sore point with the growth of the Windows market is that it increases users' expectations. Users who become accustomed to the smooth integration of off-the-shelf Windows software will have little tolerance for character-based applications. Time and again, when Windows applications are installed in client sites, the client expects that all of their software should be as easy to use as Windows.

If Windows development weren't so hard, this wouldn't be a problem. Unfortunately, programming for Windows 3.0 using procedural languages remains difficult. Using a traditional language like C, it still takes hundreds of lines of code just to put up a window.

Windows programming is difficult for two reasons:

First, there is a much larger applications program interface (API) in Windows than in DOS. Windows has over 400 calls available to the programmer for dealing with the mouse, menus, windows, dialogs, etc.

Second, Windows uses an event-driven architecture that puts the user, rather than the programmer, in charge. For example, when users run a Windows program, they can switch windows at any time, click the mouse anywhere, and generally do whatever they want. In effect, the programmer pays for the end users' ease of use.

As a result, with Windows, programmers are stuck between a rock and a hard place. Users are demanding Windows applications, but traditional tools just don't provide the necessary productivity. As a result, more and more developers are looking at alternatives, and, as it turns out, object-oriented programming is the answer to Windows' complexity.

OOP to the rescue

Since 1987, we've seen even more object-oriented development tools arise. The reason for this is that objects can very easily hide many of the complexities of programming for Windows or any other graphical user interface (GUI).

For example, in Actor, user events such as mouse clicking or menu commands are actually processed as messages to objects. As a result, there's far less control structure code required in an Actor program than in C. Similarly, the details of the API can be managed by objects themselves. Whitewater has taken this approach in a new graphics library, called ObjectGraphics, that uses graphical objects, such as pens, brushes, shapes, and coordinate spaces, to mask scores of function calls. Using ObjectGraphics, the programmer doesn't need to know anything about the underlying graphics engine.

Many of the difficult portions of programming for a graphical environment are handled very well by object-oriented techniques. Polymorphism makes it much easier to achieve a consistent user interface. Late binding makes it easy to accommodate the flexibility of a Windows interface. Encapsulation and inheritance help manage the complexity of the API. As a result, there's a growth in OOP tools across all windowing environments, including Windows 3.0, OS/2 Presentation Manager, and the Macintosh.

Asymetrix and The Whitewater Group are shipping tools for Windows 3.0, and Borland has committed to providing their object-oriented languages for Windows. Asymetrix's Toolbook, a software construction set for Windows, is closer to HyperCard for Windows than to a true object-oriented development system, but it will definitely be useful to developers who need to put a fancy front end on Windows applications.

Similarly, there are a variety of OOP tools available for OS/2 Presentation Manager, including Digitalk's Smalltalk/V PM, Choreographer, and others. The Whitewater Group has also been demonstrating the Presentation Manager version of Actor, which provides complete source code portability across environments.

And on the Macintosh, there are several OOP languages available, including OOP extensions to C and Pascal. There's also a very interesting development system called ProGraph from TGS Systems in Halifax, Nova Scotia, that provides visual object-oriented development tools. ProGraph provides a hint of what may become commonplace in the late 1990s.

OOP for GUI

The bottom line for developers today is that users will expect their applications to run in graphical user interfaces such as Windows 3.0. Character-based applications for DOS may soon have about as much appeal as those for CP/M. The most practical way to develop for graphical user interfaces is to use object-oriented development tools. Learning to use OOP for GUI applications may be the key to survival, since today's OOP tools provide the ability to compete in tomorrow's market. ■

Zack Urlocker is a technical staff member and a supervisor at The Whitewater Group, the developers of Actor, an object-oriented development environment for Microsoft Windows. Mr. Urlocker has taught object-oriented programming to hundreds of professionals and has written articles for several computer magazines and journals.

Project Management

Talking to project managers: Introducing reuse into the development process

Adele Goldberg & Kenny Rubin

As OBJECT-ORIENTED TECHNOLOGY becomes more widely employed within a software development organization, one naturally expects that software components will be reused. The goal of reuse is to develop a set of abstractions that are not limited to a single software effort. These abstractions or artifacts can be used to make future software systems development more productive while simultaneously enhancing quality.

Productivity is determined by cost and effort. In addition to the salaries of developers and testers, cost is measured in terms of the time to develop and test the software components, and the lost opportunity to do something else. Effort is measured in similar terms, plus the additional time it takes to find, evaluate, and understand reusable artifacts.

If reusing an artifact costs less and requires less effort than developing it from scratch, productivity is increased. This ability to create systems faster also means that developers can focus on the more novel aspects of the system under development. Reusable artifacts that have been proven reliable bring that history of reliability to the new system. In this month's installment of the project managers' column, we address the issues that surround the successful practice of object-oriented software reuse.

Without a doubt, one key to successful reuse is managerial commitment.

What is reuse?

Reuse is the act of using what already exists to achieve what is desired. In the context of software engineering, there are many artifacts that may be reused, including analyses, designs, implementations, test cases, and documentation.

Analyses (that is, specifications) and designs are the most abstract reusable artifacts. Abstract artifacts have limited detail and little reliance on implementation decisions. This enables their reuse during early stages of a project when they are most likely to have the largest widespread impact. Implementations are specific completed segments of code. In the context of object-oriented technology, implementations are typically individual classes or groups of related classes. Biggerstaff and others point out that the amount of a system that can be formed from reusable implementation artifacts is usually less than half the system[1]. They argue that the only way to get an order-of-magnitude improvement (in productivity and quality) is to reuse analyses and designs, rather than completed implementations. According to Biggerstaff: the higher the level of abstraction, the higher the potential payoff.

Abstract artifacts are often best created by the organization that will reuse them, since analyses and designs are representations of the business knowledge of that organization. Although abstract artifacts have the greatest potential payoff, implementation reuse is clearly the most commonly practiced form of reuse to date. Let's now examine some of the different kinds of reuse.

Categories of reuse

Where reuse occurs, it is commonly one of four types:

- Algorithms
- Classes/instances
- Application frameworks
- Interface specifications

Let's briefly look at how each is supported by object-oriented technology.

Algorithm reuse involves using the same algorithm across data structures. Typically the implementation of the algorithm is modified to handle each new data structure. Using the data abstraction supported by object-oriented technology, such an algorithm is implemented at a high level of a class hierarchy, and becomes automatically available to subclasses.

For example, in the Smalltalk-80™ system, some common searching algorithms are implemented in an abstract class called Collection which is a common parent of Set, Array, and Dictionary; some mathematical notions like absolute value are implemented in the abstract class Number and are available automatically to classes Integer and Float. New data structures (i.e., classes) can be added within a class hierarchy, and thereby inherit algorithms defined in the superclasses.

Systems such as Smalltalk-80 provide developers with application-independent reusable classes. As an example, Collections such as Set, Array, and Dictionary are reusable in a great many applications regardless of the domain. Reuse occurs in two different forms: by refining a class via inheritance to obtain a new class, and by using instances of a class in the composition of a new class. With inheritance, both the protocol and the implementation of the class are reused. With composition, just the protocol specified in the class of the instance is reused.

An application framework provides code for nearly all of the required functionality. It consists of abstract classes, the operations they implement, and the expectations for providing concrete code in subclasses that specialize the framework for particular applications. By supporting a variety of subclasses, the

© 1990 ParcPlace Systems, Inc. All rights reserved.
Smalltalk-80 is a trademark of ParcPlace Systems, Inc.

☐ *Yes,* I want to be plugged into the latest thinking and developments in object-oriented technology. Enter me immediately as a **Charter Subscriber** at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	2 years (24 issues)
☐ ~~$299~~ $249	☐ ~~$598~~ $498

outside the US — add $24 per year for air service.

☐ Check enclosed (Payable to: THE HOTLINE — US currency drawn on US bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card # _____ Exp. date _____
Signature _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____

Send Subscription Orders To: THE HOTLINE, P.O. Box 3000, Dept. HOT, Denville NJ 07834
For immediate service—call the Hotline 212-274-0640 or Fax 212-274-0646

DH90

HOTLINE on Object-Oriented Technology
The International Newsletter

588 Broadway, Suite 604
New York, NY 10012
(212) 274-0640

Do you know of a colleague who might benefit from reading THE HOTLINE? If so, please fill out below and we will promptly send them a ***free copy with your compliments.***

Send issue to:
Name: _____
Title: _____
Company: _____
Address: _____
_____
City: _____
State: _____ Zip: _____

Compliments of:
Name: _____
Title: _____
Company: _____
Address: _____
_____
City: _____
State: _____ Zip: _____

DH90S

HOTLINE on Object-Oriented Technology
The International Newsletter
The Manager's Source for Trends, Issues & Strategies

August 1990 Volume 1, Number 10

WyCASH+: An application built within an OOP environment

Paul McCullough
Nancy Deshler

US PENSION FUNDS, corporations, and banks invest billions of dollars in the "cash" markets. Cash securities are generally considered those with a remaining term to maturity of less than one year, but they can include those with maturities as long as five years. These securities include CDs, commercial paper, US Treasury bills, and short-term loans. They provide attractive yields, low risk, and high liquidity; in recent years, with the volatility in the stock and bond markets, they have grown in popularity.

While doing a consulting project for a large US pension plan, we interviewed many managers specializing in the cash markets. We heard the same complaints over and over: there are no systems designed specifically for the management of cash assets. Although some vendors actually claimed their systems could account for cash securities, managers always seemed frustrated with the system's limitations. As we explored further, we learned why.

The cash markets are incredibly diverse in nature. Securities can actually be negotiated between issuer and buyer, and new security types are frequently introduced into the market. These security types exhibit vastly different characteristics, methods of accruing interest, and payment cycles. Some earn interest at a stated annual rate; others are sold at a discount from their full face value.

For example, CDs – one of the most common cash instruments – come in a variety of different formats. Some are discounted and others are coupon-bearing. CDs with a term to maturity greater than one year generally accrue interest based on a 30/360 day count basis (assuming 12 equal months with 30 days each) and pay out interest semi-annually. Shorter-term CDs, however, accrue interest on an actual/360 day count basis and pay full interest at maturity. CDs with adjustable interest rates — "floaters" — are growing increasingly popular.

These can exhibit any combination of features: in their simplest version, they might have an interest rate reset once a quarter and will pay interest on the same day. It is not uncommon, however, to come across one that resets on the second Tuesday of the month and pays three days later!

Just keeping track of such variations is challenge enough, but all of these differences need to be accommodated by a system with accounting and analytical functionality. The cash manager needs to know when interest payments are to be received and how much the payments will be, then analyze the relative attractiveness of securities with different payment cycles. That is, a 10% interest rate on an instrument with a 30/360 accrual method and semi-annual interest payments is not at all equivalent to a 10% rate on one with interest accruing on an actual day basis and paying out only at final maturity. The manager is looking for tools to accurately model these differences.

Older systems built in procedural codes weren't designed to be flexible enough to accommodate all the possible variations. Changes needed to be introduced into the code itself, and the code was not easily adapted. Modifications and enhancements took a long time, often introducing more problems than solutions. Some managers actually reported making their investment decisions based on what their systems could and could not do, rather than the best investment decision.

One manager reported working with a mainframe-based system that could not account for any floating rate instrument. Floaters had been introduced into the market years before, but the system just hadn't been able to keep up. At the time, buying floaters was a sound investment decision managers could not make.

Similarly, investment managers need to be able to meet the changing standards of tax laws and reporting requirements. How

continued on page 3 . . .

INSIDE...

From the Editor 2
Databases 5
Mathematics and methodology
Project Management 7
Organizing for reuse
Standards 12
CALS'90 report
Dear Doctor 13
Publicity Corner 15
Product News 18
Calendar 19

FROM THE EDITOR/MARIE A. LENZI

To balance Cognos' July feature tale of gloom and doom, THE HOTLINE would like to offer an admirable success story. **Nancy Deshler** and **Paul McCullough** of Wyatt Financial Systems provide not only a Smalltalk technological success, but also success in an application area that is not as yet a hotbed of object orientation.

Nancy's focus is banking, Paul's is technology. Together, they provide a look at how objects are successfully permeating commercial and business applications areas.

After a month's hiatus, **Bob Martin** of Ontologic, Inc. continues his comparison of relational and object database management systems by comparing mathematics and methodology.

Adele Goldberg and **Kenny Rubin** of ParcPlace Systems continue their series on talking to project managers with a set of three case studies in organizing for reuse. They emphasize that reuse does not happen by accident — it is a function of both technical and nontechnical considerations, and to achieve a reasonable level of reusability, management must begin to plan for it ***now.***

Alan Purdy of Instantiations, Inc. discusses two of the initiatives of the CALS organization.

And, finally, we have been able to get **Steve Weiss**' "Dear Doctor" column started earlier than anticipated — keep those cards and letters coming!

When Nancy Deshler and I were discussing this month's feature article, she told me an interesting story: she recently attended a banking systems conference, and one presenter offered an application that was a natural for objects. When Nancy asked the presenter why an object model was not used, the response was that no one was using objects for real work. (Raised eyebrows are appropriate!)

While this is an extreme attitude, there is still a lot of "fear of the unknown" surrounding objects, particularly in the business arena; however lots of organizations are making not-so-tentative moves toward objects. I have clients coming from several different directions in development organizations (strategic planning, data administration, applications development, methodology, training...) who are asking not "what is it?" but "how do we get there?"

Over the last year or so we have watched interest and awareness in the technology grow geometrically. Conference attendees and clients are asking far more knowledgeable questions this month than they did last month. Users who were just recently becoming educated in CASE are asking about objects.

In addition, the object-oriented extensions to COBOL are beginning to gain visibility. This, of course, is juicing up the business environs with their jillion-line investment. Object-oriented technology is finally becoming accessible to them; it is finally being spoken in a language they can understand. Can objects, along with CASE, finally provide them a fighting chance to get out from under the two-year backlog and 99% maintenance mode? Or is it just another marketing gimmick?

Before we know for sure, we'll have to wait and see what the implementations of object-oriented COBOL, or COOL (COBOL Object-Oriented Language, yet another cute O-O acronym), look like. Are object-oriented extensions going to technically strengthen the language and provide more rigor and power to both application development and enhancement, or are implementations only going to pay lip service to the paradigm? Is it truly technology or is it just marketing hype? From the papers coming out of CODASYL, it's not clear to me yet whether the COBOL people will get real object orientation or only a mirage.

Marie A. Lenzi
Editor

Advisory Board

Jim Anderson
Digitalk, Inc.

Tom Atwood
Object Design, Inc.

Grady Booch
Rational

Larry Constantine
Consultant

Cort DeVoe
IBM

Adele Goldberg
ParcPlace Systems

Bertrand Meyer
Interactive Software Engineering

Reed Phillips
Knowledge Systems Corp.

Trygve Reenskaug
Taskon A/S

Bjarne Stroustrup
AT&T Labs

Dave Thomas
Object Technology International

Zack Urlocker
The Whitewater Group

Eugene Wang
Borland International

Jeff McKenna
Founding Editor

Richard P. Friedman
Group Publisher

Margherita Monck
General Manager

Elisa Varian
Production Manager

Susan Culligan
Art Director

Melanie Younossi
Marketing Manager

Gregory Tsichlis
Circulation Manager

The Hotline on Object-Oriented Technology (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, New York, NY 10012 (212) 274-0640. ©Copyright 1990 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher. Mailed First Class. Subscription rate—1 year (12 issues) $299, Foreign and Canada $323. Single copy $30.

POSTMASTER: Send address changes & subscription orders to HOOT, P.O. Box 3000, Dept HOT, Denville NJ 07834.

Submit editorial correspondence to Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280.

Versant/Nichimen

Versant Object Technology
4500 Bohannon Drive
Menlo Park, CA 94025
(415) 325-2300

Versant Object Technology announced a distribution agreement in Japan with Nichimen Corporation. Through its role as a coordinator of transnational development projects, Nichimen has earned the reputation as a global business organizer. Versant's strategic distribution channels are focused towards a global market.

Versant designs, develops, and markets VERSANT object database management systems (ODBMS). This software accelerates the design and implementation of complex workstation applications such as software development, engineering design, office automation, and computer-aided publishing. VERSANT runs on all configurations of SUN workstations and is being ported to other workstation platforms.

ObjectVision

ObjectVision Inc.
2124 Kittredge Street
Suite 118
Berkley, CA 94704
(415) 540-4889

ObjectVision Inc is shipping **ObjectVision (Release 1.0)**, a new software tool that lets users develop object-oriented programs visually. Using ObjectVision, a developer can "draw" a program's objects, interface, and database connections and then either run the application in ObjectVision or automatically convert the application into commented, ready-to-compile C++ or Turbo Pascal 5.5.

In ObjectVision, you can create new objects with a click of the mouse, then graphically add attributes and procedures. Relationships between objects can be established by drawing a line between the two objects. Working "interface objects," like buttons and switches, can be created in the built-in bitmap editor. ObjectVision also offers a number of ways to visualize and edit object hierarchies, including a "3-D view."

Calendar

THE HOTLINE CALENDAR presents conferences and meetings that focus exclusively on object-oriented technology. It should be noted that many other other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280.

July 31–Aug. 1, 1990: O-O Systems Symposium — Toronto, Canada.

Conference Chair: Larry Constantine. Tutorials, Lectures.

Contact: Digital Consulting, Inc., 6 Windsor St., Andover, MA 01810, (508)470-3870.

Sept. 25– 28, 1990: C++ At Work–'90 — Meadowlands Hilton, Secaucus, NJ. Sponsored by THE C++ REPORT & BUCEC.

The International C++ Users Group Conference. Technical sessions, tutorials, panels, BOF & exhibits.

Contact: Bob Daniels, (508)649-9731.

October 21–25, 1990: OOPSLA '90 — Fifth Annual Conference on OOP Systems, Languages & Applications — Ottawa, Canada. Cosponsored by ACM & ECOOP.

Technical sessions, lectures, BOF, tutorials, panels, and exhibits.

Contact: ACM, (212)869-7440; Conference Division.

Sept. 14–15, 1990: Symposium on OOP Emphasizing Practical Applications — Marist College, Poughkeepsie, NY.

Keynote: Bertrand Meyer. Tutorial: Richard Wiener. Technical sessions.

Contact: James Ten Eyck, Marist College, Poughkeepsie, NY 12601, (914) 471-3240 Ext. 606.

October 3–5, 1990: OOP vs. AI — San Francisco, CA. Sponsored by International Association of Knowledge Engineers.

Contact: Joan Scaffidi, (301)231-7826

November 6–9, 1990: SD–Fall — Boston, MA, Omni Parker House Hotel. Sponsored by Miller-Freeman Conferences.

Contact: Angela Hoyt, (415)267-7649.

PRODUCT NEWS

Smalltalk Apprentice Program

Knowledge Systems Corporation
114 MacKenan Drive, Suite 100
Cary, NC 27511-6446
(919) 481-4000

Knowledge Systems Corporation, specialists in object-oriented technology, now provides an innovative training program for Smalltalk, "The Smalltalk Apprentice Program," for Digitalk's Smalltalk/V for Presentation Manager. This program is a customized, project-focused, training course devoted to both developing internal Smalltalk experts and advancing the specific corporate project with which the students are tasked.

Participants are evaluated upon enrollment to determine their strengths and weaknesses. KSC's instructors then customize the course based on the participant's skill set and the needs of the project on which the participants are working. Participants are provided individual workstations in secure office space, access to KSC staff expertise, and training within the context of their project. The program is designed to run 3 to 5 weeks depending on client requirements. KSC accepts a maximum of three participants in a single session of the program to ensure optimum instructional value and a total immersion in Smalltalk via the client's project. Participants must be tasked with a project that will serve as the focus of the apprenticing or expert building process. The Smalltalk Apprentice Program builds internal Smalltalk experts by providing:

- Course content customized to the client project
- Low student–teacher ratio to encourage high information transfer
- Access to KSC development tools
- Videotape of design sessions and project demos
- Training in the key areas of:
  - Object-oriented analysis and design
  - Rapid prototyping techniques and tools
  - Maximizing use and reuse of class libraries
  - Code management and team development

Math.h++ Tools.h++

Rogue Wave
P.O. Box 2328
Corvallis, OR 97339
(503) 745-5908

Math.h++ is a library of C++ classes designed for efficient numerical computing. The library grew out of the Data Analysis and Interactive Modeling System project at the University of Washington, an attempt to bring high-level object-oriented interactive tools to the numerical modeler and data analyst. Because these were large models, efficiency and ease of maintenance were of paramount importance. Math.h++ has been utilized by a variety of users in academic and commercial settings on projects ranging from shell dynamics in mechanical engineering to ocean mixed-layer models. The library contains the following classes:

- Vectors and Matrices
- Operators and Functions
- Linear Algebra
- Statistics
- Complex Numbers
- Fast Fourier Transforms
- Slice and Pick operators
- Random Number Generator Classes

Tools.h++ is a set of C++ classes that can simplify programming. The product contains the following:

- Collection classes — a complete set modeled after Smalltalk-80 including Set, Bag, OrderedCollection, SortedCollection, Dictionary, Stack, Queue, etc.
- Generic Classes — singly and double-linked lists, stacks and queue, using <generic.h>, the current C++ approximation to parameterized types.
- B-Tree disk retrieval
- File Space Manager
- String and character manipulation
- Time- and date-handling classes

continued from page 1 ...

well they are able to keep pace with these changes and innovations has become the acid test for investment systems.

The science of investment analysis is evolving. New ways of evaluating potential investments and understanding the performance of one's own portfolio have been developed.

Although many managers might like to integrate these approaches into their investment decision-making process, often their systems just can't support such analysis. Today, Lotus 123 is still used as the last resort by many investment professionals. You have to do it yourself, and it might take a long time, but you can do almost anything you want with a spreadsheet. When these managers are spending thousands of dollars on their systems, they shouldn't have to resort to a spreadsheet. In our technological age, it should be possible to automate their analysis.

Desktop publishing on the Macintosh has raised all our expectations concerning the quality of system-generated reports and graphs. Like everyone else, investment managers want to be able to present their clients with professional-looking statements. Many take data from their investment systems and key it in to Lotus or Excel to generate their reports and graphs. Although many prefer output from the Mac, it is not viewed as a serious computer with the horsepower for the consuming calculations required for investment management. Investment managers want the best of both worlds.

We recognized a potential market. Fixed-income investment managers were dissatisfied with their systems choices. The market was there for a full-service investment management system that could support the management process from making the investment decision, through accounting for the assets, to presenting the results to clients and management.

This system would have to be flexible. More than anything else, it would have to easily adapt to changes in the industry and to managers' wish lists. It was our goal to build just such a system.

THE OBJECT-ORIENTED SOLUTION

While there are many different kinds of financial instruments, they often have common characteristics. For example, instruments selling at a discount — bankers acceptances, certificates of deposit, commercial paper, Federal Home National Mortgage Association Discount Notes — have a lot in common; because of this, instruments fit easily into the class hierarchies found in object-oriented systems.

Code specific to a particular type of instrument is found in that instrument's class, while code common to several instruments is found in a superclass. In WyCASH+, most instrument classes implement only a few methods, and most superclasses, while somewhat more complex, are still quite manageable. The following is a partial WyCASH+ instrument hierarchy, with the number of methods for each class indicated in parentheses. Class names in bold type are abstract superclasses (classes that represent common behavior and never have instances).

Instrument (95)
  Cash (21)
    Account Fee (5)
    Interest (4)
    Maturity (3)
    Principal (3)
  Futures Instrument (24)
    Eurodollar Futures Contract (6)
    US Treasury Bill Futures Contract (5)
  Security Instrument (45)
    Discount Instrument (27)
      Bankers Acceptance (5)
      Discounted Certificate of Deposit (6)
      Commercial Paper (5)
      Foreign Bankers Acceptance (6)
      Floating Interest Instrument (49)
        Coupon Bearing Instrument (49)
          Agency Instrument (6)
            Corporate Note (7)
            Floating Rate Note (6)
    Option Instrument (17)
      Option Call (2)
      Option Put (2)
  and so on...

As you can see, the number of methods to implement most instruments is small, and the hierarchy reflects the commonality of instruments.

Adding a new instrument to the system is relatively simple: determine the appropriate place in the instrument hierarchy and then define any instrument-specific behavior. Changing the behavior of an instrument is also simple: the engineer simply adds, changes, or deletes the appropriate behavior in the class of the instrument. If a change is needed to a number of related instruments, the code in the appropriate superclass is changed, and the affected classes immediately inherit the new behavior. The inheritance and modularity provided by classes makes the system easily maintainable.

In addition, because the implementation details of a class are completely hidden from the rest of the system, it is possible to completely redesign and rewrite a class (keeping the public messages the same), install the new class, and make full use of it *without changing any other parts of the system*. Another way of viewing WyCASH+ is as a framework designed to support change in the future.

Financial transactions (the buying and selling of instruments) also fall into a natural hierarchy. There is a great deal of common information — the transaction date, a reference to the instrument involved, the affected client, and so on — as well as information specific to a kind of transaction — the broker code for brokered transactions or the coupon rate for security transactions. The system again uses a class hierarchy to describe the various transactions. Extensive use is also made of class hierarchies to describe the WyCASH+'s various calculation methodologies, queries, reports, and graphs.

PLATFORM AND USER INTERFACE

We chose Smalltalk/V as our implementation vehicle because it is PC-based, has a windowing system written entirely in Smalltalk, and is easy to modify. Each developer had extensive experience in Smalltalk-80 or Smalltalk/V.

The graphical user interface is designed to support both data entry and flexible viewing of the data. During data entry, users typically interact via fill-in-the-blank fields (a typical window used for entering a financial transaction is shown in Figure 1 on page 4). Reuse of existing code is common in object-oriented software projects. WyCASH+ makes extensive use of the vendor-supplied window system, vendor "goodie" packages, and custom code.

Requests to view stored information as reports and graphs typically take the form of pointing and clicking with the mouse. Users can customize reports by adding and deleting columns, resizing column widths, subtotaling, and sorting. The format of customized reports may be saved and retrieved later. The system also supports batch printing for routine or long reports.

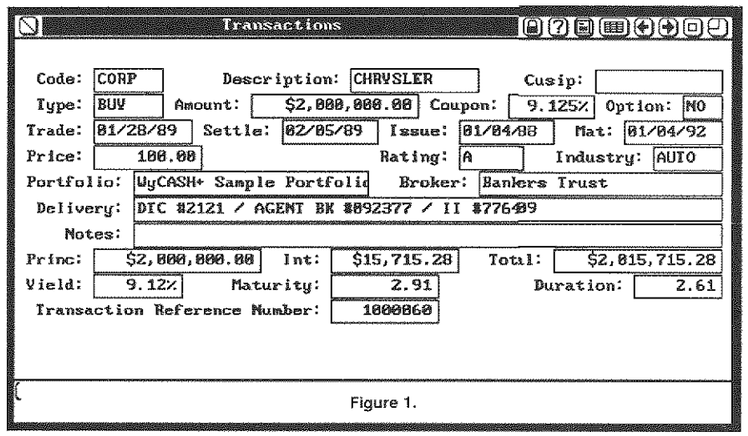


Figure 1.

DEVELOPMENT HISTORY

The development of WyCASH+ took five and a half person-years. The style of development was prototyping, and we feel that we have been quite successful at moving the prototype to an acceptable product, continuing to add new functionality. We consult extensively with three financial domain experts (nearly one per software engineer). The system is currently comprised of 750 classes.

During development, we had to overcome a number of obstacles. Moving objects from one system to another, or moving a complete financial database, has so far been handled by writing "migration files." These files are a flat representation of the objects and allow the objects to be reconstituted on another system. The migration files contain class version information, allowing considerable redefinition of object formats even within a production system.

Protecting against data loss caused by crashes (such as power failures or inadvertent software problems) has been accomplished by writing transactions to a similarly formatted log file so they may be reapplied during recovery.

Currently, the system's database is maintained as Smalltalk objects in RAM. While this is acceptable for low- to mid-volume usage, financial houses with a high volume of transactions will eventually begin running out of storage space for objects; obviously, some sort of database is needed to provide a repository for the objects.

Our development style has been to make changes to the representations of classes as needed, writing simple routines to migrate data from old formats to new as needed. Merging this flexibility with the relative rigidity of a database system remains a challenge, and we are currently evaluating the various options.

To test new instruments and detect inadvertent changes to existing instruments, we have an extensive testing system. Tests are developed using Excel spreadsheets, and those test results are imported to the WyCASH+ test harness. WyCASH compares its results to those in the spreadsheets and reports discrepancies.

Some clients have requested a multiuser version of the system; because the system is highly modular, we were able to produce the desired system in less than six weeks.

THE SUCCESS

WyCASH+ was officially released in January 1990. In literally every demonstration of the software, prospective users are amazed. The system has yet to make a bad first impression. The challenge, of course, is living up to that impression.

Prospects generally seem impressed with the interface and the wide range of features the system offers, but, most importantly, when given the opportunity, we have nearly always been able to impress them with our responsiveness.

For example, one prospective client who saw the system was very interested, but questioned one of our calculations. There are two typical ways of computing a statistic. We did one; they preferred the other. In three days, we were able to include the method they preferred and take the system back to show them! No other vendor can do that.

We do meet some resistance due to the look of WyCASH+. With its graphical interface, it just doesn't look like the investment system managers are used to. Many of our prospects have never used a mouse! WyCASH+ looks to them more like a toy than a serious software system. But such objections are easily overcome by just a few minutes of hands-on experience with the software.

Our biggest challenge now is living up to our own expectations for the ongoing development of the system. We've set high standards for ourselves, but, so far, WyCASH+ has been able to live up to them. ■

Nancy Deshler is a product manager for Wyatt Software, responsible for the design and development of the WyCASH+system. Prior to joining Wyatt, Deshler was Assistant Vice President and Portfolio Manager for the Patterson Capital Corporation, a fixed-income investment advisory firm in Los Angeles. She graduated Phi Beta Kappa from the University of Southern California, where she earned her BA and MA.

Paul McCullough is a senior software engineer at Wyatt Software. Prior to this, McCullough was a founder and owner of Instantiations, an object-oriented consulting firm. His original object-oriented experience was gained in five years at Xerox PARC as a researcher. McCullough holds a BS from California Polytechnic University.

("relational," "Micro Channel Architecture," etc.) it's going to be used more as an abbreviation for "good" and "buy this" than for its real meaning.

Source: "'Object-Oriented': A Buzzword that Brings Confusion, Benefits," **Washington Business Journal,** Scott D. Palmer, May 21, 1990.

DBMS: Information Builder position papers talk about object-orientation and admit they don't mean the strict classic definition of object-orientation. What is being referred to then? Are you just talking about the screen objects under the upcoming PM version of Focus? ...

Cohen: Oh, the graphic product will do lots of manipulating of graphic objects. But graphic objects are a different notion than database objects or programming objects...

DBMS: Right, I understand that, but I see them shamelessly mixed up in the literature, and I'm ...

Cohen: Terrible, terrible — and unfortunately, we're just starting to use the same terminology that everybody else is using. PM Focus does have a very object-oriented screen handling system. But our truly object-oriented Focus will come later when we switch from a client server to an object-server environment...

Source: "The Future of Focus," **DBMS,** Interview with Gerald Cohen, May 1990.

STANDARDS

Digital Equipment has joined the Object Management Group ... One OMG official said DEC had asked that its membership be kept secret until June 12, when the vendor said it would be making a substantial but unspecified object-oriented announcement. The 75-company OMG is working to define a reference model and object model for object-oriented technology. ... OMG also seeks to define interfaces to object databases; a distributed API; access to common services such as mail, spell-check system and network management; and document and data architectures. Major members include Sun Microsystems, Hewlett-Packard, Nixdorf, Servio Logic, Object Sciences, Bull, Philips, and Data General

Source: "DEC Joins the Object Management Group," **UNIX Today,** April 30, 1990.

Christopher Stone, president of the Object Management Group (OMG), Westboro, Mass., envisions a time when the end-user will be able to buy software as easily as he or she buys components for a stereo set, without having to worry about operating systems or hardware platforms. ... To realize this vision, OMG is promoting acceptance of object-oriented standards. OMB compliance will be a kind of Good Housekeeping seal of approval. End-users will know that any product that carries the seal will work with any other product. OMG was formed in April 1989 and promises to develop a set of standards for object-oriented compliance by this summer.

Source: "Software Metamorphosis," **VAR Business,** Mark Kindley, March 1990.

... As Chris Stone [director of OMG] points out, "A commonly accepted definition of object still hasn't been hammered out. In addition, object management standards are still years away."

Source: "25 Tough Integration Problems and Solutions! Object-Oriented Programming," **Systems Integration,** April 1990.

Citing the need for a standard framework to build applications with artificial intelligence, five major companies representing users, hardware manufactures, and software vendors have formed a coalition to develop an AI shell. ... The five companies are Ford Motor Co., U.S. West, Texas Instruments inc., Digital Equipment Corp., and the Carnegie Group, Inc. ... The Carnegie Group will act as the lead developer working on the representation component of the system, which handles how knowledge is represented by object-oriented programming. [The group will be known as] the Initiative for Managing Knowledge Assets [IMKA). ... The technology is expected to run on as many different platforms as possible, with UNIX to be the backbone.

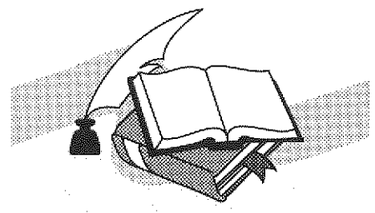
Sources: "Five Major Companies Team to Develop AI Shell," **MIS Week,** Kim Ann Zimmerman, April 30, 1990; "AI Initiative," **UNIX Today,** April 30, 1990.

"If you build the specifications from an interface, you allow the market to decide who has the best implementation," [Christopher] Stone said. Unlike Goldberg [Adele, ParcPlace Systems], he believes OOP standards are necessary now. ... But some at the [Seybold Technology Forum] wondered if it isn't too soon to set OOP standards. "How can we set standards when we don't know the answers to questions like, 'What objects are there?' 'What classes are there?' and 'What concepts are there?' said Edwards [John, Associative Design Technology.]

Source: "Forum Examines Object-Oriented Design," **UNIX Today,** Mike Azzara, April 16, 1990.

Object-oriented design is today's "in" technology, yet, in the words of Ken Orr [principal, The Ken Orr Institute]: "In a sense, object-oriented languages constitute a programming solution in search of a front-end methodology." ... There is no standard object-oriented database language in which to program applications. Indeed, there is not standard data model on which to base such a language.

Source: "Solving Software Problems," **System Development,** Philip J. Kiviat, March 1990.



orientation to meet our needs through the 1990s?

Source: "DBMS: Next Wave Could Be an Easier Ride," **Data Processing Digest,** May 1990.

NEWSWORTHY REVELATIONS

Eiffel ... will be put into the public domain, according to Bertrand Meyer, President of Interactive Software Engineering, Inc. of Santa Monica, Calif., and developer of the language. By relinquishing the trademark and creating a consortium of users, Meyer sees better prospects all around.

Source: "Eiffel Put in the Public Domain," **Unix World,** May 1990.

A growing demand for UNIX employees is appearing on the manpower side of US technical support positions and systems administrators. ... When it comes to application development in UNIX, the ... areas in demand are knowledge of X Windows, graphical user interfaces, object-oriented programming, object-oriented languages, and communications and networking.

Source: "The Expanding UNIX Job Market," UNIX World, Lynn Haber, May 1990.

The single biggest threat to the future of the CIO is that, while the key requirement for the position is an understanding of the elements that drive the business, a pending technology revolution driven by the acceptance of CASE and object-oriented programming tools could overwhelm the CIO who is too business- oriented and not knowledgeable about the technology.

Source: "MIS Experiences Darwinian Revolution," **MIS Week,** Leslie Goff, May 21, 1990.

LARGE COMPANIES SUPPORTING OBJECT-ORIENTED TECHNOLOGY

Steve Jobs got a much needed endorsement from, of all companies, IBM, when the computer giant announced plans to offer NeXTStep on high-end PS/2s and on their new line of workstations running AIX. NeXTStep is NeXT Computer Systems' object-oriented user interface and development environment. ... Big Blue's support positions NeXT as a worthy and legitimate environment for business and professional productivity applications.

Source: "NeXT News," **The Sloane Report,** April 1990.

New Wave is used by HP ... in its office automation package, New Wave Office. It is also used by Data General Corp., Westborough, Mass., in its CEO Object Office and by AT&T in its Rhapsody office automation package. Because all use New Wave, all can interoperate. ... New Wave is also supported as a standard by the Object Management Group, Framingham, Mass., which is working to formulate standards for object-oriented software.

Source: "HP Unveils New Wave," **Communications Week,** Stanley Gibson, May 28, 1990.

"... the next steps are object-oriented programming applications to be created with standard tools and open networking that can interface with proprietary-type networks to provide an OSI [Open Systems Interconnection]-type environment." [Gilbert Williamson, President, NCR.]

Source: "NCR Has a General Purpose: No Niches," **Computer Systems News,** Mark Mehler, May 21, 1990.

AT&T Computer Systems hopes that its new groupware system, ... Rhapsody, will be music to users' ears. The package is said to automate and facilitate ad hoc and routine group tasks like project management, multimedia electronic messaging and monthly results tracking. ... AT&T used the object-oriented technology of a major rival — Hewlett-Packard Co.'s New Wave — in the product.

Source: "AT&T Offers a Groupware Rhapsody," **Software Magazine,** May 1990.

ANTI-HYPE

The ongoing argument about "true" and "hybrid" [OOP] systems is a good illustration of fuzziness in action. In February 1990 **Computer Language**, Scott Ladd justly chastised the rash labeling of products claiming the new magic object-oriented ingredient. ... He describes object-oriented programming in terms of support for four minimal features: data abstraction, encapsulation, inheritance and polymorphism.

... Anything less than this disqualifies; anything more is window dressing. ... Without a digression into the meaning of these four reasonably unambiguous concepts, at this stage in the evolution of object-oriented design, Ladd's "definition," while better than the loose, anything-goes approach we both dislike, is overly restrictive.

Source: "Bit by Bit: Grand GUIgnol," **Computer Language,** Stan Kelly-Bootle, May 1990, pp. 149–152.

Loy [Patrick ...] quotes a remarkable prophecy by Tim Rentsch in 1982: "My guess is that object-oriented programming will be in the 1980s what structured programming was in the 1970s. Everyone will be in favor of it. Every manufacturer will promote its products as supporting it. Every manager will pay lip service to it. Every programmer will practice it (differently). And no one will know just what it is."

Source: "Bit by Bit: Grand GUIgnol," **Computer Language,** Stan Kelly-Bootle, May 1990, pp. 149–152.

The hottest new buzzword in computing these days is "object-oriented," and you can expect to see it applied to everything from PC screen displays to programming languages. Like other buzzwords,

≡≡ DATABASES

Mathematics and methodology

Robert R. Martin

As a spokesman for a leading object database company, I am often asked if object databases have a firm mathematical foundation, as is the case with relational databases. The short answer is "Yes." The longer answer is that, because object systems address a much broader spectrum of software development than the relational model, the mathematical treatment of object systems is not as simple as the relational algebra.

In reading this article, it is important to understand that I am a technical manager, not a theorist. My background is in building distributed, data-intensive applications, not in database theory or mathematics. Therefore, rather than focusing on details of the models, I will outline a manager's perspective on the significance of the underlying mathematics of both object and relational models.

THE RELATIONAL MODEL

There is a formal underlying theory of relational systems, based, in part, on set theory. Its major aspects, as outlined by Codd and Date, are:

The relational abstraction

Data structure:

Domains (the set of values an attribute can assume)

Relations (attributes, tuples)

Data integrity:

Entity integrity (via non-null primary keys)

Referential integrity (foreign keys match primary keys)

Data manipulation:

Relational algebra — select, project, product, union, intersect, difference, join, divide

Relational assignment

This model was first published by Codd in 1970 and is based in the mathematical works of Whitehead and Russell, published in 1910. In this model, all data is defined in terms of tuples (records), and is manipulated with a small set of relational operators. This mathematical model has been highly promoted as a key factor in the acceptance of relational databases.

The relational model is a big improvement over earlier CODASYL and Network models. First, it is an abstract interface to data, not a physical organization. Relational databases can be reorganized without changing applications programs (data independence). The industry has standardized on a common data definition and manipulation language, SQL. A key virtue of the model is its simplicity; designers can learn the algebra in a short period of time.

RELATIONAL MODEL LIMITATIONS

The good news is that the model is simple. The bad news is that it is too simple, forcing programmers to work at too low a level of abstraction. Real-world applications are rarely so simple! Programmers have to translate complex, real-world models into the relational model. The gap between the structure of the real world and the structure of the solution is called the semantic gap. Crossing this wide gap is a long, slow, error-prone process, and it is unaided by the relational mathematics!

An analogy exists in the computer hardware world. There is a basic integrated circuit, called a NAND gate, that has a simple, mathematical model. By combining NAND gates, all possible computer circuits can be built. This corresponds to applying relational operators. Yet no manager would ever ask a designer to sit down and build a computer from NAND gates. It's just the wrong level of abstraction. Designers use higher-level building blocks, such as memories, registers, and ALUs, to build the system. In electronic design, multiple levels of abstraction, layering, and composition are used extensively.

In a relational database, all data structures must be converted to relations (NAND gates). High-level application abstractions, such as documents or purchase orders, must be decomposed into this basic, low-level table abstraction. Even basic, well-proven structures such as linked lists, trees, arrays, and lattices must be converted to relations. This often obscures the programmer's intent, results in poor performance, and increases programming and maintenance costs.

Much is made in relational theory of the notion of essentiality, of reducing the model to the minimum data and operations required to manage information. In my opinion, this is an area where we have let theory carry on a bit too far! This is like saying your calculator should not have multiply and divide keys because you can always use addition and subtraction to perform these operations!

This minimalist philosophy is contrary to trends in programming languages and expert systems, which are moving towards the increased expressiveness and richer modeling power offered by techniques such as inheritance and composition.

In short, the relational model is good as far as it goes, but it simply does not go far enough. The mathematics are interesting, but not particularly helpful to the programmer.

Screen, reports, transactions, and concurrency control fall outside of the math-

ematics, yet are used by most database applications. The bulk of data manipulation in typical data-intensive language is performed inside the host programming language, not within SQL. Yet no one worries about the mathematical basis behind C, Fortran, or Cobol, as it is simply too difficult to prove mathematical correctness in large programs. From my perspective, the relational mathematics fails to address the primary problems in software design and maintenance. The math works, but it ignores the hard problems!

THE OBJECT MODEL

Object databases are built upon the advances in data independence and query abilities pioneered by the relational model. Unlike the relational model, object databases integrate an object programming language with database capabilities. This integration provides a rich, expressive modeling capability. The major components of the model are listed in Figure 1.

The object model, as embodied in languages such as C++, is less rigorously defined and more complex than the relational model. It is, however, far more expressive and includes the critical software concepts of inheritance, layers of abstraction, and composition.

Object databases are unifying advances in programming languages (objects, abstraction) with those in data management (data independence, integrity, query, transactions). Clearly, an object database presents a higher level of abstraction to the programmer and significantly narrows the semantic gap.

We have been working with several object experts, including David Childs and John Edwards, to more fully articulate the underlying mathematics of the object model. In speaking with them in preparation for this article, I learned that objects have a very rich heritage in both math and physics.

Edwards developed a powerful core object model that draws upon elements of class and functional calculus (see Figure 2).

In addition to the model, Edwards has developed a methodology for object-based design and has a graphical modeling tool, called Ptech, which supports the methodology. This tool can interpretively execute the models so you can test your assumptions. In addition, it can generate C++ and object database code directly from the models to produce production applications.

As a development manager, I can really get excited about this. It takes the concept of single paradigm programming and extends it to cover the entire software process, from conceptual modeling through to implementation and maintenance, significantly reducing the semantic gap.

It also applies a sound, mathematical foundation to the area of software design where it has the most leverage — in conceptual modeling. If you don't get the concepts correct to start, you can be sure that your applications and databases will be wrong!

CONCLUSIONS

So what does all of this mean to a software manager? First, relational databases have a sound mathematical basis, but at the expense of a restrictive, low-level abstraction, forcing you to program around it. Object models are more complex, but much more expressive.

Finally, more rigorous object models are now being applied to analysis and design in the form of object methodologies. Combining an object methodology with object languages and databases results in a simple, uniform, high-productivity programming environment. ■

Robert Martin, a founder and Vice President of Products for Ontologic, Inc., has lead the commercial introduction of object database technology. Ontologic is the leading supplier of second-generation object database systems. For more information, contact Ontologic in Burlington, Mass., at (617)272-7110.

Abstract object model

Data structure:
- Abstract Data Types (extension of Domains)
- Classes (similar to relations)
- Aggregation (Sets, Lists, Arrays, etc.)
- Extensibility
- Inheritance
- Composition
- Polymorphism

Data integrity:
- Entity (via system-supplied unique IDs)
- Referential (system enforced)
- Structural (enforcement of arbitrary rules)

Data manipulation:
- Relational, Class and Functional algebra/calculus, as embodied in object-oriented DMLs such as C++, Object SQL, Smalltalk or CLOS.

Figure 1.

Core object model

Concepts
Corresponds to a real-world concept. You know you have a concept when you know when it does and when it does not. For example, ***marriage.***

Class
The extension of a concept, or all objects to which a concept applies. ***All married people.***

Object
Corresponds to a real-world object. An object has a state, which is the set of all concepts that apply to the object. ***A person.***

Event
A change in the state of an object. ***Get divorced.***

Process
A lattice of events. ***Get married, then divorced, then married again.***

Figure 2.

PUBLICITY CORNER

Clippings Summary

The number of clippings totaled 115: 45 articles in the monthlies, 4 in the bimonthlies, 49 in the weeklies, 9 in the biweeklies, 7 in the dailies, and 1 quarterly. Approximately 33% were product announcements or reviews; 25% were about o-o programming, languages or tools; 12% were marketing news or hype, 12% were about O-O applications; and the remainder was comprised of general technical articles, editorials, and articles on graphics.

APPLICATIONS

Standards researchers also may create some leading-edge GIS (Geographical Information Systems) applications...using expert systems and object-oriented technology developed at NOAA's Nautical Charting Research and Development Lab under artificial intelligence program manager Dave Pendleton.

... Object-oriented technology contributes an ability to capture lore in the form of production rules — tens of thousands of them, from detail-level rules all the way up to general rules, Pendleton said. "The way we use objects is to define classes of things, in this case charts with channels in them, which are made up of buoys and soundings and lights," he added. ... A buoy is part of the channel, which is part of the harbor, so the buoy inherits the rules that were defined for the channel. In the same way, the channel inherits rules defined for the harbor, and the harbor inherits rules defined for its geographic area.

... He added that any agency that uses Earth-coordinate data for geography, economics, weather, fisheries or forests would benefit using a knowledge-based GIS.

Source: "Geographic Information Systems," **Federal Computer Week,** Cheryl Pellerin, May 14, 1990.

The fascinating thing about object-oriented tools is the potential they have for providing LAN services. Using object-oriented development techniques, the trick is to reduce the essence of the LAN kernel to the most fundamental file system and memory management elements. All other elements become objects that the kernel processes.

Source: "LAN-centricity and Application Development Tools," **LAN Technology,** Steve Papermaster and Steve Guengerich, May 1990, pp. 16–17.

Object-oriented DBMSs are beginning to draw serious attention, but major questions have to be answered before IS organizations embrace the technology wholeheartedly. The absence of standards and of reliable ways to evaluate performance claims made by object-oriented DBMS vendors are two critical barriers to wide-spread commercial implementation.

... "For MIS executives, the issue is whether — and how — they want to manage yet another data model that is incompatible with the relational model," says Robbe Burnstine, an analyst with the Gartner Group Inc.'s office in Acton, Mass.

... A reputation for poor performance continues to dog object-oriented databases. "It's an unfair rap," says Jack Orenstien, a member of the technical staff with Object Design, Inc., a vendor of object-oriented databases in Burlington, Mass. "Requirements addressed by object-oriented DBMSs are very different from those addressed by relational databases," he says.

Source: "Taking On Relational," **Information Week,** John Soat and Daniel Todd, May 21, 1990, pp. 69.

Some observers predict that the products that succeed will be those that combine object-orientation with relational database capabilities. "Object-oriented database is very much at the stage where relational was seven or eight years ago," comments Shaku Atre, partner at Atre/Computer Assistance, a consulting firm in Rye, New York. "... object-oriented is not going to replace relational, it's going to be added on to relational." ...

Object-orientation, according to Atre, will enable the banking industry's databases — thus far geared primarily toward numerical data — to store voice and image information as well.

Source: "Capitalizing on Databases," **Bank Systems & Technology,** Kenneth Silber, May 1990, pp. 42–46.

One user believes the success of either group of vendors [relational vs. object-oriented DBMSs] will depend on its ability to market products based on benefits to the user instead of the technology. In the meantime, users are faced with both an opportunity and some difficult decisions when considering an object-oriented DBMS.

They will have to answer three questions: (1) What are the DBMS capabilities we will need to gain a competitive advantage through our information systems strategy? (2) What organizational changes need to be made to take fullest advantage of the end-user-empowering, object-oriented technologies? (3) Are we partnering with a DBMS vendor that has the technology, complementary software and strategy

Doctor feels that those who do not learn from history are doomed to repeat its mistakes. History has demonstrated that technical problem solving requires that the solvers first characterize the problem (analysis or requirements definition) and only then think about a solution.

Further, history shows that a problem solution requires statement of the solution (a blueprint or design) before the solution is actually asserted (construction or implementation). The analysis–design– implementation gradient is as important now as it was in the past.

Actually, it's more important on projects of any significant size. In the past, the two fundamental units of code organization were the granular line of code and the aggregate function or subroutine. Object orientation has three: the retained line of code and method and added object, a larger aggregate that packages related data and process. As a result, we are finding that it is not difficult to be successful on smaller problems in an object-oriented environment without any formal methods. It is easy to be unsuccessful on the larger problems.

A reasonable approach, then, is to adopt an evolutionary system development method embracing object-oriented system organization and use it reliably.

Many have been published. Synthesis, developed by The Doctor and Meilir Page-Jones, was discussed in THE HOTLINE in December 1989. Other methods have been proposed by Bullman, Coad, and Yourdon and Shlaer and Mellor. Although The Doctor is, of course, predisposed toward one of these over the others, he is of the strong opinion that any method, even a weak one, is better than none at all.

The Doctor feels that the method must address both function and data organization. Both are crucial in an object organized system. He will be pleased to discuss the characteristics of a well-formed object-oriented method on request.

Dear Doctor:
Can I do object-oriented programming in FORTRAN?

The Doctor has also been asked this question about COBOL, C, Ada, and even assembler. In all cases, the answer is a qualified yes. The degree of objectification, however, does depend on the language you use.

Technically, one writes object-oriented programs in an object-oriented language that implements encapsulation, inheritance, and possibly dynamic binding (there is some debate about the latter). Actor, C, C++, Eiffel, Objective-C, and Smalltalk fit into this category.

Some languages, however, only provide packaging mechanisms. In effect, they provide the machinery for encapsulation but not inheritance. Such languages, Ada for example, can be called object-based. They have some, but not all, of the features of an object-oriented language. A user of an object-based language can expect to get some, but not all, of the benefits of an object-oriented language.

Finally, some languages don't provide any help at all, FORTRAN and COBOL for example. Even in such environments it is still possible to gain some of the benefits of objectification if the designer and programmer discipline themselves. Such object-structured designs and code make heavy use of information clusters or data abstractions, which are a form of home-grown encapsulation. As with object-based environments, inheritance is not implemented or supported directly.

There are, then, three degrees of objectification. First is object-oriented. It requires an object-oriented language and provides encapsulation, classes and inheritance.

The next is object-based. An object-based environment provides some of the features of object-oriented environments but not all: encapsulation and user-defined data types.

Last is object-structured, which provides encapsulation only. It requires designer and programmer discipline because fewer facilities for encapsulation are provided by the environment. The problems associated with building object-structured systems in FORTRAN are not dissimilar to the problems of writing structured code in assembler. It's possible. The Doctor thinks that it takes work, talent, and a more than casual understanding of the object abstraction.

The Doctor has been a consulting methodologist for over a decade. He invites you to submit your questions on object-oriented technology to:

Dear Doctor
1202 108th Avenue East
Puyallup, WA 98372

≡≡ PROJECT MANAGEMENT

Talking to project managers: Organizing for reuse

Adele Goldberg & Kenny Rubin

There are but two ways of rising in the world: either by one's own industry or by profiting by the foolishness of others.
— La Bruyéré

In the July issue of THE HOTLINE, our article outlined the definitional and organizational assumptions behind the reuse of software components. That column explained why reuse will be beneficial to you and why you should start now to plan for reuse. Our thesis was that developing reusable software components results in both cost and productivity benefits, but only if the process, the team, and the reward structures are appropriately planned.

The substance of this month's article is an illustration, through examples, of how various object-oriented projects have incorporated reuse. The intent is to empirically study issues surrounding reuse that have influenced the success or failure of a project.

The case studies that we have selected are actual development efforts undertaken by organizations willing to be interviewed by us. All of the programs involved were written using object-oriented languages.

To begin, let's review the high-level reuse process that we outlined in July:

- Acquire reusable artifacts through purchase or development.
- Classify and document the artifacts according to the organization's engineering practices.
- Find, understand, and use artifacts sanctioned by the organization.

© 1990 ParcPlace Systems.

This process spans two areas of responsibility, one by management and the other by developers. Managers must adopt a software engineering process that recognizes and supports reuse. This includes a component acquisition plan, management of the developers with respect to the adopted process, and review and reward structures commensurate with developer reuse activities. Developers must overcome personal biases against reusing code written by others; they must learn and practice the techniques for proper development and utilization of reusable components.

A goal of both managers and developers is the creation and proper maintenance of a library of reusable components. Building up such a library may occur through acquisition and customization of third-party software components or by in-house development.

Either way, both developers and managers must set up and adhere to procedures whereby reusable artifacts meet the organization's development standards and can be effectively incorporated into additional software development projects.

This requires a decision process by which artifacts are entered into a shared library, including how the artifacts are classified and conventions for their documentation. There might be different libraries — for example, one for the corporation as a whole, another for a department or project team, and another for the individual. Each library would have different controls, signifying the extent of corporate-wide acceptance of the contents.

With respect to building up a corporation's library, we have one key bias — we do not believe that the solution to successful acquisition of reusable artifacts is solely through the purchase of commercial packages. Therefore, we did not attempt in our interviews to create a catalog or taxonomy of commercial packages you should acquire. If we had been guided by the much abused comparison of software components with hardware components, such a commercial market would have dominated our recommendations for acquiring corporate software libraries.

The revolution created by reuse of hardware components is a poor argument for software reuse. Software components typically can be and are customized, whereas hardware components rarely are. The library of reusable components supported by an organization is a statement by that organization of how information and processes are to be codified.

Although the library may have been derived from commercial software packages, those packages are then customized to suit the needs of the organization better. In fact, it is best to buy an initial specification and an open implementation, available for modification. This customization is in definite contrast to the reuse of hardware components, which are used precisely as purchased.

Moreover, it is typically not feasible for a consumer who needs a hardware component to manufacture that component. Software is remarkably different. Often, the consumer is a professional programmer, capable of creating the software component. Strong motivation must exist if this programmer is to reuse software created by someone else.

The difference between reuse of software and hardware components shows up legally as well in that the external specification of the software is the clever design that is copyrighted. For hardware, the

proprietary implementation and manufacturing process are more significant.

By contrast, implementation details of software can be openly shared. External software specifications, which of course have to be shared, represent the important aspect to be valued (priced) and legally protected.

In doing the case studies, we looked for sources of reusable components, ways of storing and accessing reuse artifacts, mechanisms for setting up the libraries, ways in which to organize the project team, and changed expectations built up from the availability of reusable artifacts.

CASE STUDIES

We conducted seven case-study interviews. They covered projects involving from one programmer to teams of eight — in one case the project was divided among three teams of six to eight programmers. They involved creating prototypes as well as products. Some relied solely on in-house development; others included contract development. We have selected three of these cases to discuss here: a prototype that evolved into a product, a throwaway prototype, and a product now in the maintenance stage. For each of the examples, we provide a description along the following dimensions:

- Project description: what is the project, and what is its lineage?
- Managerial expectations: did management encourage/reward reuse?
- Development team composition: was a team member assigned the role of reuse tester or librarian?
- Access to reusable components: how were reusable artifacts obtained and managed?
- Development lifecycle: how did reuse affect the lifecycle?
- Outcome: was the software successfully completed? did reuse aid or hinder development?

Case study 1 — ➲ a CASE environment

Project Description. The purpose of this project was to create a prototype of a new information–analysis environment. The idea was to map the information to be analyzed onto directed graphs and then use a particular user interface structure for traversing the information space.

A prior project created a new approach to interacting with the complexities of a computer operating system and the administration of that system, basically a new user interface application framework. The general idea was to create a way to interact with computer-accessible information such that the user would have direct visual and physical means of moving through information structures to access a wide variety of content stored within those structures. This mapped directly to the kind of user interface that this CASE project required.

Managerial expectations. The managers of this project expected the developers to begin by creating a proof-of-concept prototype that demonstrated the functional ideas for the CASE environment. The prototype would be shown to the targeted market to solicit feedback. Based on this feedback, a decision would be made either to develop a full production system or to terminate the project.

The developers were expected to heavily reuse analyses, designs, implementations and test case components that were available to them. Since first-phase developer evaluation was based on having a working demo in approximately two months, the developers were motivated to reuse. There was, however, no explicit performance–evaluation model based on numbers of old components reused and new components generated.

Development team composition. The team began as one programmer who developed the fully functional prototype. After positive market feedback, a product team was formed, including several programmers and an even larger crew of testers and documentors.

Access to reusable components. The organization in which the work was done supported a significant library of reusable components that were sold on the commercial market. These were naturally available to the CASE product team. In addition, a prior project had created an approach for interacting with visual information.

The code from this prior project was not sanctioned by the organization, i.e., it had not gone through product-level design review nor formal documentation, and hence was not officially available as reusable code. It was, however, known to exist and was openly accessible on a shared file server. Because of the tight schedule and the insistence of the project manager that these user interface components were necessary for a timely completion, the code was reused in the new project.

The component framework that was reused was modified to address the needs of the CASE environment more directly. The new, modified components were classified according to the needs of the project and entered into the project's own repository. There was, however, no designated librarian, neither for the project nor for the larger organization. Neither the original code nor the modified versions were independently documented and submitted for a design review.

Development lifecycle. The organization did not have enforced guidelines for documentation, nor did it have a required sequence of prototyping and design reviews. Reasonable engineering practices were expected, but the manager of this project chose a unique approach, isolating this project's work from that of the remainder of the organization in the interest of personal control and development speed. Design documents were prepared only after the product was delivered. The project suffered from two problems: the analysis prototype was used as the basis for the final product, and the prototype was built with components that had not been tested and accepted for reuse.

Outcome. The project was completed with approximately half of the system based on reused implementation code. But this statistic is not relevant in the face of what turned out to be an inadequate result. Reuse had a very negative impact on this project, but only in the sense that it was poorly regulated. Because of the reusable code and the object-oriented tools available to him, the project manager reported that he simply found it too easy to create a very complex result. The ease of development, the ability

CALS AND OBJECTS

But what does CALS have to do with objects? Several of the aerospace speakers displayed graphs showing how their companies organize their information bases. Not surprisingly, the information base is at both the core of the company and the core of the computing infrastructure. Mediating between the information and other processes is a layer annotated as the information model. In a few companies (e.g., Boeing), this model layer talks about the information contained in the core in terms of objects. They use objects to describe information because objects are more expressive than other alternatives.

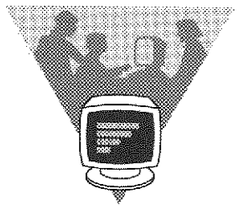
None of the installed information systems are deployed using object-oriented components or languages. Information managers in the manufacturing industry plan to replace their existing relational base, but only when object database technology stabilizes and major vendors begin to support object database products.

PDES will also serve as a major testbed for object technology. While PDES is today a research effort funded by a consortium of aerospace manufacturers, the funding companies are fully committed to evolving a set of standards using object-oriented languages for the description and definition of mechanical components.

In summary, the CALS mission of radically reducing the cost of military hardware will greatly reduce the cost of all design, manufacturing, and support processes. Objects, object languages, and object databases will be key to making the CALS mission possible. This is especially true when these companies capture the enterprise-wide information required for concurrent engineering. ■

Alan Purdy is President and Co-founder of Instantiations, which develops and supplies services and technologies to the object-oriented marketplace. Prior to this, Purdy was amember of the research staff at Xerox PARC, where he lead a team in integrating database technology with object-oriented languages. He was co-founder of Servio Logic and chief architect of Servio's GemStone object server. In 1986, Purdy served as co-chair of ACM's First Conference on Object-Oriented Systems, Languages and Applications. He can be reached at (503)242-0725.

≡≡ DEAR DOCTOR



A forum on object-oriented methods, practices, and procedures.

Dr. Steven Weiss attending.

Dear Doctor:
We have been using structured methods for some time. Now, with object-oriented methods becoming popular, we are concerned that we will need either to abandon our existing techniques or to avoid using object-oriented techniques altogether. We don't consider either of the choices to be acceptable.

Is the decision that simple? Are there ways to make use of our knowledge of structured methods to migrate to object-oriented methods?

It appears that there are two object-oriented schools: the evolutionists and the revolutionists. The evolutionists acknowledge that the ideas and principles of object orientation are demonstrably derived from the existing attitudes, procedures, and methods that preceded them, almost the way birds were shown to be demonstrably derived from their preceding reptilian ancestors.

The revolutionists, on the other hand, maintain that object orientation is a brand new ground-breaking discipline that has no significant ties with the past. Although both views have merit, The Doctor has more confidence in the evolutionist's interpretation.

These days, software engineering disciplines don't generally arise by spontaneous generation. They usually have antecedents. The relevant antecedents of object orientation include function (or process) orientation, which represents a system by focusing on the actions that it performs and data orientation, which represents a system by focusing on its underlying data structure. Each attempts to look at a system from a unique, and somewhat isolated, point of view.

The object-oriented view contributes significantly by combining the process and the data views (encapsulation), declaring them both in a single component (the object) and treating them as fundamentally indivisible. Under the object-oriented paradigm, information from the function-oriented view is reflected primarily in methods and information from the data-oriented view is reflected primarily in instance variables.

That, then, is a site that has some methods background to do. The

STANDARDS

CALS Expo-89 Conference

Alan Purdy

I was only vaguely aware of the Computer-aided Acquisition and Logistic Support (CALS) initiative before attending the CALS Expo'89 conference in Orlando, Florida, December 1989. The CALS phenomenon and the conference were a pleasant surprise: while CALS initially focused on the military–industrial sector, its more profound effect will be seen by commercial manufacturers. In addition, it will provide a full-scale test of object languages and databases. This article describes CALS and its objectives, reports on this attendee's perceptions of the conference, and discusses the relationship between CALS and the growing object industry.

WHAT IS CALS?

In September 1985, Deputy Secretary of Defense William Taft launched a project known as the CALS initiative. In Taft's words, CALS is "a strategy for transitioning from our current paper-intensive system to a largely automated and integrated mode of operation ... with substantial progress by the end of the decade." The product of this initiative is a set of standards and specifications that define formats, structures, and protocols for exchanging information.

The initial phase of CALS produced a set of standards for the exchange and storage of text and graphical documentation. The removal of paper represents an important challenge for the military and industry. For example, each Navy ship maintains approximately 15 tons of service and repair manuals. One observer noted that if all the paper manuals were removed from the Navy, each ship would float four inches higher in the water.

Many word-processing systems can already read and write documents in a Standard Generalized Markup Language (SGML) format. Soon, all defense contracts will require SGML compliance for internal and external documents, radically reducing the cost of communicating across distance and time and removing a major barrier separating people.

In the second phase, CALS will develop standards for exchanging engineering, design, and manufacturing information in support of the entire lifecycle of a product.

If you examine the process of designing, manufacturing, and supporting a major engineering product (say, a Boeing 747), you begin to understand the domain this second phase addresses. Such products often have a lifecycle of 30–50 years. If you doubt this, recall that one leg of our strategic defense triad consists mostly of the venerable B52, first delivered in 1954.

Additional metrics help illustrate the problems faced in this domain. A Boeing 747 contains over a million parts, from thousands of vendors; its user and repair manuals weigh several tons; thousands of designers at hundreds of sites cooperate in the design of a new airframe. Similar problems confront designers and manufacturers of all machines, from army tanks to VCRs.

The efforts to automate the design process have gone through several stages. At the first stage, structural designers used computers to simulate how proposed designs would behave under static loads. Along with finite element analysis, additional programs evolved to simulate other aspects of prospective designs. However, these early simulation programs (islands of automation) could not communicate directly with each other.

Next, programs appeared to convert the output of one simulation "island" to a form readable by the next island in the design process. Over time, these systems evolved into integrated CAD systems, bringing all the steps of the design process into one seamless system. Manufacturing process engineers experienced a similar evolution, starting first with Bill of Materials programs (BOM), then to Computer-Aided Manufacturing (CAM) and Manufacturing Resource Planning (MRP). Today, Computer-Integrated Manufacturing (CIM) promises to integrate all the islands in the manufacturing process.

At each stage in the integration of these islands, the main barrier has been lack of standards for information exchange. While nearly all computer vendors support the lower layers of the ISO 7 level model, the higher layers (application, session) have only begun to stabilize.

Many of the CAD systems on the market today comply with Initial Graphics Exchange Specification (IGES). A consortium of users, CAD vendors, and National Institute of Standards and Technology (NIST), led by CALS, is evolving the replacement for IGES: Product Data Exchange Specification (PDES). I will discuss this more in the next section.

Concurrent engineering (CE) is at the horizon of today's manufacturing vision. CE is a computer-aided environment allowing subsystem and component designers to interact with other designers and with manufacturing process engineers toward the same goal: to produce a working product at the lowest cost. CE systems span company boundaries and time, integrating historical information with details of the product to be constructed. CALS addresses this vision by coordinating and focusing the research required prior to the creation of standards.

CALS'89 CONFERENCE

Over 2000 people from manufacturing, aerospace, and the computer industry attended the CALS conference. The trade show had 150 vendors demonstrating CALS-related hardware, software, and communications gear. The attendees heard a common message — Michael Hallman, then president of Boeing Computer Services, said it best: *"CALS should not be embraced because it is mandated by the government but because it represents sound business practice."*

Several speakers reviewed the progress of CALS during the last five years, listing IGES, SGML, and growing vendor participation among the primary achievements. Others issued words of caution: we have a long way to go before we actually replace all our existing paper.

In summary, the conference attendees, vendors, and speakers were all enthusiastic participants and supporters of the CALS initiative.

ity to incrementally add complexity, the speed with which a novel result was obtained — all contributed to a fast development cycle void of reasonable reviews. This project clearly highlighted to us the need for discipline in how software components are shared within an organization. It is also a reminder that object-oriented technology must be introduced within the context of existing engineering practices.

Case study 2 — ➲ a decision support system

Project description. The objective of this project was to design a proof-of-concept for a decision support system consisting of a variety of novel ways to access business-related information. Assuming a large and growing database about business documents and business processes, the basic idea was to present the user with a computer-based information center to retrieve documents, examine processes, and generally ask "what if?" questions.

This system, a proof-of-concept prototype, was the first of its kind for a domain in which the developing organization had substantial expertise.

Managerial expectations. The managers expected the developers to rapidly design and demonstrate a prototype system to a very large potential customer. The desired end result was a contract with the potential customer to build a full-scale implementation.

Because they were in a hurry, the managers not only encouraged but required the use of a variety of both internal and third-party object libraries. This managerial mandate for reuse was based on the organization's successful reuse practice on many of its previous non-object-oriented systems. Specifically, over the years this organization had developed substantial expertise in designing and developing many transaction-oriented systems using COBOL. In doing so, they had also constructed and validated cost/benefit models that incorporated the reuse of a variety of artifacts ranging from analyses and designs to implementations and test cases. What was lacking, however, was a rigorous cost/benefit model that would enable project planning to take place at the same level of detail and confidence as was possible in the COBOL environment. To some extent, this prototype was a test case for developing such a model.

Development team composition. Two programmers were assigned to the prototyping task. The implementation took approximately 20 person-weeks (from early February to early April). One person, who had an extensive programming background, began learning the Smalltalk-80 language in December the year before.

The second person had no programming background, and began learning Smalltalk-80 in February. Both programmers took a formal training class in which the concept of reuse was emphasized. A documentor was added toward the end of the project to create a user guide for the target client.

Access to reusable components. The prototyping effort was leveraged off of three sources for reusable components. The first source was the rich library of basic data structures and visual objects provided with the Smalltalk-80 system. The second source was a toolkit of reusable user interface gauges that was purchased from a third-party vendor. The gauges in the toolkit are designed to be domain independent. The team reused the gauges without trying to understand their implementation.

The third source of reusable components was an existing library developed by the organization itself. This library contained domain-independent user interface widgetry that had been developed during earlier protoyping efforts.

The components in the library were available to projects developed at the same company installation; they have not been made available throughout the corporation. All of the components were released by individuals to other individuals, with no formal corporate process.

Because the numbers of individuals were in the tens, not hundreds, this approach was proving satisfactory for the short term. A senior manager had developed some of the components, and remains a key advocate of reuse in the organization.

> ***... successful reuse on large projects will require formal cost/benefit models and explicit policies promoting reuse.***

The organization modified the standard Smalltalk-80 development tools to help find components in their library. For example, additional query options were added to some of the System Browser menus to facilitate locating special kinds of components. As for understanding and utilizing components, the organization relied on the substantial cross-referencing and codifying tools already present in the Smalltalk-80 system.

Development lifecycle. This project did not originate with a firm requirements specification. The intent of the project was to impress a potential client, so direct client involvement in the development of an analysis prototype was not appropriate. Furthermore, since the goal was a mock-up prototype (i.e., one that allows the

user to sit down and drive the system without the real background machinery in place), the most important aspect was the design of the user interface. Hence, the analysis phase consisted of understanding the various types of information to be manipulated and the navigational metaphor for logically moving through the information space.

During the construction of the analysis prototype, the organization was able to reuse existing user interface technology (i.e., hypermedia-based navigation) that it had developed for previous prototypes. The reuse of this technology, including its design and implementation, was the primary factor that enabled the organization to mock-up a prototype consisting of very novel visual effects in a short time period.

On the other hand, because the extent of this project was limited to the construction of an analysis prototype, the developers were not expected to devise any new reusable components. Management was knowledgeable enough to recognize that reuse does not happen by accident, but rather is a planned engineering event.

Since the purpose of the project was to develop a demonstrable prototype without regard to the underlying architecture, management had no misconceptions about the possibility of generating new reusable components at this stage of development.

Outcome. The goal of the prototype was to demonstrate to a prospective client that a new kind of system could be developed that would suit their business needs. When the prototype was shown to the client, they were impressed enough to award a major contract to the organization to develop a full-fledged system.

The application developers felt their success had much to do with the fact that a collection of reusable components was made available to them. Empowered by these components and a powerful set of tools, they were able to meet their deadline. This is truly a reuse success story.

Case study 3 — ⊃ an information management system

Project description. The purpose of this project was to create a general-purpose information management system that included some basic office automation functionality. It included a sophisticated spreadsheet, as well as document-creation and forms-generation packages. At the onset of this project, there were no prior object-oriented applications on which to base development.

Managerial expectations. The managers of this project expected the end result to be a deliverable product that would address the needs of the client. With respect to reuse, the managers neither expected nor enforced any formal reuse policy. In part, this is because this project was begun at a time when there was not much awareness of the issues surrounding reuse. Today, this attitude continues in part because the system is now in maintenance, not development. For new projects, management feels that the developers have demonstrated their team competence in achieving required goals in a timely fashion, and so there is no need for change. We interpret this to mean that the developers can choose whether or not to reuse existing designs, frameworks, or implementations.

Since reuse was not specified as a project requirement, nor as a managerial goal, the organization had no formal review or compensation strategy that involved either the generation or reuse of project artifacts. Rather, review and evaluation were based primarily on timely delivery and resulting client satisfaction. The high level of mutual cooperation within the group allowed the team to meet these goals.

Development team composition. The development team consisted of six to eight professional programmers, primarily experienced Smalltalk developers. One of the programmers was the system integrator. She became a *de facto* librarian, who could have easily helped coordinate rewrites for better reuse. Our understanding is that information about available components, decisions to abstract reusable components from components created by independent programmers, and even assignments of components to be built were done in hallway conversation, but not in a formal process.

Although the developers claimed that many reusable components were developed during this project, no reuse testing was carried out. In our June column, we defined reuse testing as the act of testing potentially reusable components in new situations to determine the extent to which they are reusable. Reuse testing is a precondition to entering components into a corporate repository. This group had no such repository and, as we stated earlier, no formal process for identifying and testing reusable components.

Access to reusable components. This project was based on the commercial library that is packaged with the Smalltalk-80 development system. In addition, the developers generated many of their own reusable components. These new reusable components exist at various levels of abstraction. Some of these are:

- Utility methods, such as a help facility.
- User interface components, including specially designed dialog boxes (developed over a two-year period).
- Packages for generic end-user applications, such as an image editor or a forms editing and generation package.

During the development stages of this project, one factor that influenced reuse was the inherent limitedness of the development and execution platform. The small memory and low processor speed promoted reuse by motivating a reduction in the number of components to be used. Such reductions presumably saved space and increased speed because there were fewer objects to be managed.

Although many different reusable components were generated, there was no motivation, either through manager's mandate or developer's desire, to package these components in a form that would permit their delivery to the rest of the corporation. To be sure, some of these components were reused on additional projects, but always by the same group that developed them. Moreover, there was little turnover in the group. Without external users or new group members, there was no motivation to make components available for wider reuse.

To the extent that there was information on potentially reusable components, it occurred as a by-product of generating a manual about the system code. To support the generation of such manuals, the team developed a special "manual writer" tool.

In addition to formatting documentation about the software, this manual writer collected and reported statistics about the implementation, including the number of each kind of object, class hierarchies, and method counts. Since the team often relied on the generated documentation to learn about the software, the existence of this tool encouraged the developers to provide general-purpose comments about how the software components they developed were to be used.

Development lifecycle. Although management did not concern itself with the issues surrounding reuse, the developers themselves installed and practiced an informal reuse process. The team has worked together for several years, meeting often through "hallway encounters." Team members have developed a mutual trust of one another's technical abilities. This trust eliminated many of the social inhibitors for successful reuse of one another's code. In general, however, reuse did not change the established software development lifecycle within the organization.

Outcome. The project was deemed a success based on perceived user satisfaction. However, there are other dimensions to success, such as how easily the system is modified to include enhancements and bug fixes. And, more important from our point of view, how much of the application can be reused the next time a similar system needs to be built? The developers of the system feel that they have created a repository of reusable implementation artifacts that can be reused on the next similar project. However, in the interview, we could not obtain a clear description of these reusable components. The reusable artifacts are catalogued in the heads of the developers, and might not easily be extracted by new personnel who join the team.

To further enhance the success of this team, we believe that more attention should be given to the reuse process. The team has been successful in the past because they are bright and motivated and have been working together on the same system for several years. If some of these factors were to change, it's not clear how to assure comparable success.

SUMMARY

This column examined how several specific object-oriented development efforts dealt with reuse. Although our sample space is quite small, some interesting similarities seem evident. First and foremost is the lack of a formal cost/benefit model for helping project managers make decisions about reuse. Without such a model, it is difficult for managers to generate cost estimates and schedules. Of all the projects that we examined, not one had a schedule that inspired managerial confidence.

In addition, there are no clearly defined metrics with which developers can be evaluated during performance reviews. Measures such as the number of components reused (where available) and the number of new reusable components generated should be a part of any review and evaluation strategy. However, in the projects we examined, there was no explicit a priori understanding between managers and developers as to how reuse would factor into the evaluation process.

Developers did, however, know that evaluations took into account timely completion and user satisfaction. This motivated developers to reuse existing components in several cases.

Along another dimension, none of the companies we visited had formal corporate repositories of components, other than the one that sold commercial packages. Even in this case, there was no formal way in which components not part of those packages were managed within the company. Rather, components that were potentially reusable often were located in a particular individual's personal file-system directory. Likewise, no project had a person whose job was that of a librarian (as we defined this job in the June issue of THE HOTLINE).

How is it then that some of the projects were deemed successful (from a reuse perspective) if these things didn't exist? We feel the answer is project team size. All of the projects surveyed were of small to modest size (i.e., from one person to eight persons). When the number of developers is small, the reuse processes can be successful when performed in an ad hoc manner. For example, to reuse a component, a team member must be aware of its existence and be able to access it unobtrusively. On small teams, a developer can usually find components and avoid stepping on anyone's toes simply by talking to the other developers. On teams where there are a large number of developers, it becomes geometrically more difficult to coordinate such reuse activities. We believe that larger projects would have less probability of success unless:

1. there are formal procedures for informing all developers of available artifacts;
2. the artifacts are stored in a manner that provides multiple and concurrent access, following consistent conventions for documentation; and
3. there is a mechanism for maintaining revisions and derived versions of reusable artifacts.

In addition, successful reuse on large projects will require formal cost/benefit models and an explicit review and evaluation policy that promotes reuse. ▮▮

Dr. Adele Goldberg is President and CEO of ParcPlace Systems. Goldberg spent 14 years as researcher and manager at Xerox Palo Alto Research Center. She has served as president of the ACM and has received the ACM Systems Software Award.

Kenny Rubin is a computer scientist at ParcPlace systems, where he has been a primary curriculum developer of three of the ParcPlace Systems Smalltalk-80 training courses. In addition, he has served as lead instructor for many of the ParcPlace Systems courses taught over the past two years and has written on object-oriented technology, artificial intelligence, and human–computer interaction.

HOTLINE on Object-Oriented Technology
The International Newsletter

Charter Subscription Form

☐ **Yes,** I want to be plugged into the latest thinking and developments in object-oriented technology. Enter me immediately as a *Charter Subscriber* at the discounted term marked below and rush me the current issue. This is a risk-free offer. I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues)	*2 years (24 issues)*
☐ $249	☐ $498

outside the US — add $24 per year for air service.

☐ Check enclosed (Payable to: THE HOTLINE — US currency drawn on US bank)
☐ Bill me: att _____
☐ Purchase order enclosed
☐ Charge my ☐ Visa ☐ Mastercard

Card # _____ Exp. date _____
Signature _____

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Country _____ Phone _____

Send Subscription Orders To: THE HOTLINE, P.O. Box 3000, Dept. HOT, Denville NJ 07834
For imediate service — call the Hotline 212-274-0640 or Fax 212-274-0646

DOJD

HOTLINE on Object-Oriented Technology
The International Newsletter

588 Broadway, Suite 604
New York, NY 10012
(212) 274-0640

Do you know of a colleague who might benefit from reading THE HOTLINE? If so, please fill out below and we will promptly send them a *free copy with your compliments.*

Send issue to:
Name: _____
Title: _____
Phone: _____
Company: _____
Address: _____
City: _____
State: _____ Zip: _____
Country: _____

Compliments of:
Name: _____
Title: _____
Phone: _____
Company: _____
Address: _____
City: _____
State: _____ Zip: _____
Country: _____

DOJDS

HOTLINE on Object-Oriented Technology
The International Newsletter

The Manager's Source for Trends, Issues & Strategies

September 1990 Volume 1, Number 11

Introducing OOPS through design – it's all in the words

Seth Copen Goldstein

A successful programming project begins with understanding the user's terminology, wants, and needs. In the same way, the successful introduction of object-oriented programming requires an awareness of an MIS shop's vocabulary, wants, and needs. Just as we ask users to define why they want the system, we must ask why people are introducing OOP into their shops. The most common answer is to increase productivity. But, like most users, they don't want to pay the price — even if it is going to amortize itself quickly.

How does one go about bringing object-oriented concepts into a firm? The choice becomes obvious if we look at the problem from an O-O viewpoint. The way to successfully introduce object orientation (a phrase I hate, but one that is amorphous enough to embody design, programming, and thought processes) is with a combination of object-oriented design and traditional programming. For those of you who are already convinced that this process begins with O-O design, I send the message "skip the next eight paragraphs."

CHOOSING THE TEST PROJECT

In addition to choosing the point at which to introduce object orientation (at design or programming time), a correct choice for the project is required. Answers to this ubiquitous question range from implementing a toy example in an object-oriented language or reimplementing an already completed project to implementing a large project from scratch. These possibilities ignore either the facets of O-O programming that make it a productivity improvement, the politics of introducing a new technology into a company, or the primary lessons of O-O technology.

I propose that no matter what kind of project is chosen (and I will show later why a toy example or a reimplementation of an old project is doomed to failure), beginning with an O-O programming language will ensure defeat. If O-O programming is proposed, an immediate echo of *"Which language?"* will be heard.

I won't attempt to touch on the fiery topic of which O-O language is the best. The debate will begin with *"Which language is the standard?"* and will continue to *"Where can we find programmers proficient in the language?"*

As the controversy heats up, people who believe in the O-O buzzword will argue the benefits of each language — without any real experience in which concepts are most beneficial or even necessary for the project, among them: encapsulation (which all can agree on), inheritance (single or multiple), binding (dynamic or static), runtime or compile-time (aren't those the same?), message-passing (isn't that just function calls?), typing (strong or weak; how is that different from binding?), and language model (hybrid or pure).

On the other side of the coin, the really proficient programmers on the team will be

continued on page 4 ...

Inside...

From the Editor 2
Down Under 3
The Australian object-oriented scene
Review 7
Borland's "Objects on tour"
Analysis & Design 8
Design principles for modular product development
Management 12
Asset management and O-O technology
Book Review 14
OBJECT ORIENTED DESIGN — WITH APPLICATIONS
Dear Doctor 15
Publicity Corner 17
Product News 20
Calendar 23

FROM THE EDITOR/MARIE A. LENZI

All too often we hear companies say we can't just stop doing procedural development and begin building objects. In this month's feature article, **Seth Goldstein** of Complete Computer Corporation discusses a way to "ease" into object-oriented development by *Introducing OOPS through design* — an approach that introduces objects at the analysis and design level and allows for either procedural or object-oriented implementation.

In *Design principles for modular product development*, **Doug Bennett** of Design Ways discusses the problems and practices of our current approach to analysis and design, suggesting alternative priorities that can modify the development process.

At SCOOP Europe in July, I made a concerted effort to establish a network of European contributors. The first entry from this effort is *Asset management and object-oriented technology* by **Trevor Hopkins** and **Brian Warboys** of Manchester, England. They discuss the concept of object libraries as assets, much like hardware and buildings. They also discuss what it takes to get there — and to stay there. This notion of assets needs to be taken seriously. *Very seriously!* It is truly one of the main drivers of the development and growth of reusable parts. Or should it be the other way around? Also internationally, **Dr. Brian Henderson-Sellers** of Kensington, New South Wales, discusses *The Australian scene*.

THE HOTLINE has two reviews this month: I take a look at Borland's OOP World Tour and **Bob Howard** of The Whitewater Group reviews *OBJECT ORIENTED DESIGN — WITH APPLICATIONS* by Grady Booch. Bob is one of the few people I know who develops software using object-oriented languages while also using and teaching structured techniques and methodologies.

The issue of managing and overseeing object libraries has begun to get much attention. Now that we're beginning to develop libraries of reusable parts, people have started to speculate on the management and control of these libraries. The one (big) thing in our favor is that we're asking the right questions before it's too late to domesticate the beast.

Now, to get it right, we need to describe the task fully, to staff it intelligently, to give it the power it needs to be effective, and, finally, to name it properly.

The task is, generally, to create and manage an organization's store of reusable software parts. More specifically to:

- Identify, review, build, catalog and administrate classes.
- Oversee projects to ensure that potentially reusable parts are identified, existing parts are reworked for reusability, parts are not abusively reused (requirements are not skewed to reuse a part), and parts are not rebuilt (the "*not invented here*" culture still lives).
- Know the contents of the library. Development teams can eat up a lot of precious time looking for reusable parts, particularly if compensation is tied to reuse and not to lines of code. This may not be a large issue now, but it will become so as more libraries become available for purchase. (Sophisticated tools are going to be required, and soon.)
- Participate in the development and maintenance of the Enterprise Wide Object Model. This task is critical. Object Library Managers must work closely with the strategic planners. There needs to be an interdependence between the two functions, but autonomy is required. One group cannot serve the other and remain effective.

Staff it with a combination of developers and managers who are technically skilled, are managerially wise, can think broadly, can work on immediate "problems" with a perspective on long-term implications, and can work either in teams or independently (wouldn't we all like a staff like this?). The point is, this is not exclusively a technical effort nor can it take place in isolation. Object library managers

MARIE A. LENZI
Editor

Advisory Board

JIM ANDERSON
Digitalk, Inc.

GRADY BOOCH
Rational

LARRY CONSTANTINE
Consultant

ADELE GOLDBERG
ParcPlace Systems

MARY E. S. LOOMIS
Versant Object Technology

BERTRAND MEYER
Interactive Software Engineering

REED PHILLIPS
Knowledge Systems Corp.

TRYGVE REENSKAUG
Taskon A/S

BJARNE STROUSTRUP
AT&T Bell Labs

DAVE THOMAS
Object Technology International

ZACK URLOCKER
Borland International

STEVEN WEISS
Wayland Systems

JEFF MCKENNA
Founding Editor

RICHARD P. FRIEDMAN
Group Publisher

MARGHERITA MONCK
General Manager

ELISA VARIAN
Production Manager

SUSAN CULLIGAN
Art Director

MELANIE YOUNOSSI
Marketing Manager

GREGORY TSICHLIS
Circulation Manager

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, New York, NY 10012 (212) 274-0640. ©Copyright 1990 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher. Mailed First Class. Subscription rate—1 year (12 issues) $299. Foreign and Canada $323. Single copy $25.

POSTMASTER: Send address changes & subscription orders to THE HOTLINE, P.O. Box 3000, Dept HOT, Denville NJ 07834. Submit editorial correspondence to Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280 (212)274-0640; Fax: (212)274-0646.

New Books John Wiley & Sons, Inc. announces **REUSABILITY AND SOFTWARE CONSTRUCTION IN C AND C++** by Jerry D. Smith, which covers the importance of object-oriented programming for software reusability, hierarchical dispatching mechanisms, notification-based programming, and methods from minimizing the use of global variables in coding without producing functions burdened by long parameter lists. The book is priced at $32.95. Contact: Wiley at 212-850-6000. (Source: **Cadence**, June 1990.)

C++ PRIMER, Stanley B. Lippman, Addison-Wesley; 617-944-3700; 464 pages; paperbound.

PROGRAMMING IN C++, Stephen C. Dewhurst and Kathy T. Stark, Prentice-Hall; 201-592-2000; 233 pages; paperbound.

WHAT IS C++? Narrated by William M. Miller, Software Development Technologies; 508-443-5779; 101 minutes; VHS cassette.

Calendar

The Hotline Calendar presents conferences and meetings that focus exclusively on object-oriented technology. It should be noted that many other other conferences are including sessions on the technology. To have a meeting or conference listed, please send the information to Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280.

Sept. 14–15, 1990: Symposium on OOP Emphasizing Practical Applications — Marist College, Poughkeepsie, NY.

Keynote: Bertrand Meyer. Tutorial: Richard Wiener. Technical sessions.

Contact: James Ten Eyck, Marist College, Poughkeepsie, NY 12601, (914) 471-3240 Ext. 606.

October 3–5, 1990: OOP vs. AI — San Francisco, CA. Sponsored by International Association of Knowledge Engineers.

Contact: Joan Scaffidi, (301)231-7826

October 21–25, 1990: OOPSLA '90 — Fifth Annual Conference on OOP Systems, Languages & Applications — Ottawa, Canada. Cosponsored by ACM & ECOOP.

Technical sessions, lectures, BOF, tutorials, panels, and exhibits.

Contact: ACM, (212)869-7440; Conference Division.

Sept. 25– 28, 1990: C++ At Work–'90 — Meadowlands Hilton, Secaucus, NJ. Sponsored by The C++ Report & BUCEC.

The International C++ Users Group Conference. Technical sessions, tutorials, panels, BOF & exhibits.

Contact: Bob Daniels, (508)649-9731.

October 18–19, 1990: World of O-O Applications — Amsterdam, the Netherlands. Sponsored by Object Management Group and Patricia Seybold's Office Computing Group.

Emphasis on applications and solutions as a means of exploring Object Technology and its business benefits.

Contact: SPOCG, (617)742-5200

November 6–9, 1990: SD–Fall — Boston, MA, Omni Parker House Hotel. Sponsored by Miller&Freeman Conferences.

Contact: Angela Hoyt, (415)267-7649.

GeoDOS Graphical Multitasking Environment

GeoWorks Inc.

PC Week (June 18, 1990, "Graphical GeoDOS to give Low-End PCs Multitasking") reports that "GeoWorks Inc. this fall plans to release 'GeoDOS,' a graphical multitasking environment that—unlike Microsoft Windows and OS/2 Presentation Manager (PM)—can run on PCs with an 8088 processor, 512K bytes of RAM and Hercules monochrome graphics hardware, sources familiar with the product said last week. ...

"The system reportedly will also include an object-oriented interface-building tool, similar to NeXT Inc.'s NeXTStep, sources said. The retail package that GeoWorks plans to deliver by November will offer an integrated set of office applications, including a full featured word processor, a graphics package and a communications program, as well as more than a dozen scalable outline fonts."

Turbo C++

Borland International
408 438-8400

Dr. Dobb's Journal ("MIDI, Turbo C++, Token Pasting, and PC Hot Keys," June 1990) reports that "The news this month is Turbo C++. Following [its] success with object-oriented Pascal 5.5, Borland has assaulted the C++ object-oriented market with guns drawn and blazing. Originally intended to be an adjunct to Turbo C 3.0, the C++ face of Borland's compiler is now at its forefront. By emphasizing the object-oriented side of their most popular compiler product, Borland has made a policy commitment to object-oriented programming as the paradigmatic wave of tomorrow."

Mac Hypermedia Tool on the PC

Spinnaker Software Corp.
617 494-1200.

PC Week ("Spinnaker Brings Mac Hypermedia Tools to the PC," June 11, 1990) reports that "Spinnaker Software Corp.'s new OS/2 Presentation Manager and Windows 3.0 versions of its hypermedia programming environment may prove to be an effective vehicle for porting Macintosh Hypertext applications to the PC, according to beta testers.

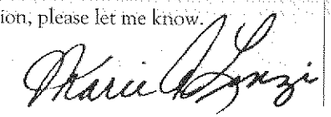
"Written in HyperTalk — Apple Computer Inc.'s object-oriented HyperCard programming language — the OS/2 PM and Windows 3.0 versions of Spinnaker Plus will allow users to build hypermedia applications that integrate text, graphics, sound and video, according to Wolfgang Kitzka, the creator of Spinnaker Plus."

need to be working on projects with the developers, users and strategic planners.

With responsibility must come control. The people who are doing this job must have the power to enforce their work — to give people responsibility without control is just not cost effective. Watching one's work become relegated exclusively to documentation is frustrating; when this occurs, enthusiasm for the job diminishes exponentially. There will be no library of reusable parts if we don't let the people responsible for the work do what they must to get the job done. Part of the power comes with proper placement in the organization. The function may need to be placed fairly high on the organization chart, possibly reporting directly (or almost directly) to the CIO or Director. Keep in mind, I'm talking about a position of significant corporate responsibility, not the "Object Police."

It's imperative the function be named properly. **Larry Constantine** does a teamwork workshop in which the first thing the team does is establish an identity by picking a name and a logo. This denotes, both textually and graphically, who the team is and what they do. The title that has been used of late is *librarian*. But librarian is only one responsibility, and it's not number one on the list. Librarian, to me, connotes a quiet, reserved, passive function. We need a name that suggests something more active and assertive.

Administrator is another name I've heard, but the first thing that comes to mind is Data Administration. The people who create and manage the library of reusable parts are responsible not only for the data, but also for the functions that transform it. Administrator is too passive a title as well. I've been using the title *Object Library Manager*, but I'm not altogether pleased with that name. Manager is so overused it means almost nothing. If anyone out there has a better title for the function, please let me know.



≡≡DOWN UNDER

The Australian object-oriented scene

Dr. B. Henderson-Sellers

An object-oriented Australia is beginning to emerge. Enthusiasm is increasing day by day. In Melbourne and Sydney, Object-Oriented Special Interest Groups have been established within the last few months, both under the auspices of the Australian Computer Society. At the Victorian SIG in Melbourne, Bertrand Meyer was the speaker at the May meeting, while in New South Wales the inaugural meeting in Sydney coincided with the launch of Ontologic's OODBMS in Australia. Educational companies, often in collaboration with other local industries or universities, are establishing reputations in professional O-O training. This should be encouraged by new legislation, effective July 1, 1990, requiring all companies with an annual national payroll of greater than $200,000 to spend at least 1% of their payroll per annum (rising to 1.5% in July 1993) on accredited training courses.

Other recent events of note were the fifth Australian Software Engineering Conference (ASWEC), held in Sydney in May. While covering the whole gamut of software engineering, there was an obvious emphasis on O-O. The first (keynote) paper by Bertrand Meyer was followed by papers on object-oriented lifecycle development methodologies from the University of New South Wales, several papers on formal specification, Object-Z from the University of Queensland, and a contribution of software validation from the University of Oulu in Finland. The undercurrent of object orientation continued throughout the conference, and in the preconference tutorial sessions one of the four half-day seminars was, for the first time, on object-oriented development. Again in Sydney, the three-day presentation by Larry Constantine and colleagues (both from the United States and homegrown from Oz), *"The Object-Oriented Systems Symposium,"* was attended by well over 100 information systems/computing professionals. The sessions included overview presentations and tutorials together with some shorter, more specific lectures. Although it's increasingly evident that commercial organizations are beginning to utilize O-O ideas, many don't wish this to be disclosed, as they believe strongly that the use of O-O technology will give them a competitive advantage.

There are many other O-O presentations being made in Australia. I was pleased to be invited to address the Australian Institute of System Analysts in Sydney in June; others are making in-house presentations based on standard papers and texts. At the University of New South Wales, we realize the importance of exposing both undergraduate and postgraduate students, if only briefly as yet, to the new ideas of O-O techniques; and employers are asking, of the universities, when they can expect to be able to employ graduates with O-O skills.

These are exciting times to be involved in object-oriented software engineering. ■■

Dr. B. Henderson-Sellers is a senior lecturer in the School of Information Systems at the University of New South Wales in Kensington, New South Wales, Australia.

continued from page 1 ...

asserting that they already do object-oriented programming, so who needs a new language? Finally, upper management, the people funding the project and thus the ones that really need to be convinced, will be wondering about the viability of committing resources to an untried technology.

Even if the above controversy didn't take place, introducing object orientation by reimplementing a project or coding up a toy example would be imprudent. The benefits of O-O programming are mainly found in programming in the large. If a small enough project is found to satisfy upper management as to its limited risk, it will either not show any real benefits to satisfy anyone; or any visible benefits will be attributed to the unreal smallness of the project. What will be observed is the learning curve of the programmers in relation to the new syntax.

In the same way, reimplementing an old project with a new O-O language will not endear anyone to the O-O languages. If the project is being picked for a rehash, it has either not been successfully executed in the first place or it is being heavily used and requires substantial modification.

In the first case, the team will either be fighting an uphill battle against the project's bad press or will be bored because they have already done it once before. In the second case, the project will expand beyond a test implementation as users "require" the new version to include all the items missing on the first go-round. Thus, those in charge will view the result as "just a test," while the programmers are really creating a new product. The result can hardly be flattering to the project team or the O-O language picked. If the team does meet with success, it will be attributed to the fact that the project had been done once already.

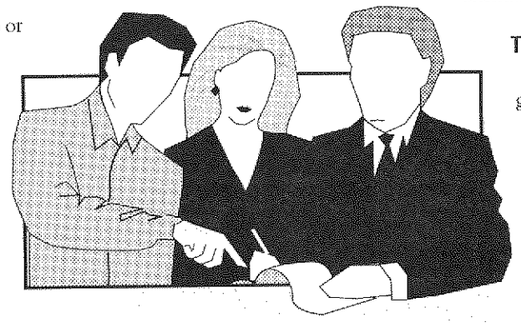
Politics aside, the main reason each of these approaches fails is that the programmers will never have a chance to do the most important part of any project based on O-O concepts: picking the objects. If a lifesized project is chosen and the focus is on an O-O language, then unless the design accounts for objects, the object-oriented implementation will be unnatural.

If the language chosen is "pure" (e.g., Smalltalk), the resulting code will be muddled, because an improper set of objects was chosen as the base for the implementation. If a "hybrid" language (e.g., Complete C) is chosen, the programmers will in all likelihood disregard the object-orientedness of the language. The end result, in both cases, either will not be object oriented or will be unmaintainable, unreadable, and probably unusable.

Arguments about which language to choose aside, the biggest barrier to the success of introducing object orientation is not the syntax of a language, nor is it the conceptual difficulty or newness of encapsulation or inheritance, but rather the vocabulary of the O-O community. When talking about O-O languages and design, we use words referring to concepts that have been in use for years. This confuses the issue, as well as trivializes it. Listeners feel there is nothing new except for obfuscating vocabulary. (The first word that needs to be redefined is the word "OOPS." Certainly, no one can take seriously a technology that sounds like a mistake.)

However, after using O-O design, they can see the value of the O-O part of the process. In a good environment, they even make the connection as to the value of implementing the project with something that supports O-O concepts directly, whether an O-O DBMS or an O-O language.

As the author of COMPLETE C, I once thought the best way to introduce O-O concepts was through an O-O language. I believed O-O design couldn't effectively be used without an O-O language. It is only after doing the design (in an O-O manner) for several projects where the implementation language was non-object-oriented, ranging from relational database projects in Oracle to straight C implementations, that I could see the tremendous value in doing O-O design — regardless of final medium.



THE RIGHT QUESTION

Thus, the appropriate question to ask to ensure success of the project is not what language, or what project, or even what kind of project, but rather *what are the objects in the project?* This primary question is answered by an object-oriented approach to design. If this approach is taken in the analysis and design, it will naturally aid the implementation, whether in an O-O or traditional language.

Once we realize that introducing object orientation with programming is fated for failure, the natural question is where to start. The answer is obvious: at the beginning. After the basic idea for the project has been formulated and the scope of the project loosely defined, a strategic study is begun. As this phase is often combined with analysis, we will treat the two phases as one (without getting into arguments over the details of software engineering, even if separated, the strategy study and analysis contain many of the same steps and deliverables, but at different levels of detail). It is here that object orientation can be easily introduced.

THE ANALYSIS PHASE

Successful O-O design/programming is based on finding and defining the objects in the problem domain. As with any other programming method, users are consulted, and a specification is written using the words the user understands to define the system to be built. Then, and

Objectworks from Smalltalk-80/ Windows 3.0

Parc Place Systems Inc.
415 691-6700

Infoworld ("Smalltalk-80 for Windows Ready to Ship in Third Quarter," June 4, 1990) reports that ..."Joining the growing ranks of object-oriented programming (OOP) systems for Windows, Parc Place Systems Inc. said it will ship in the third quarter Objectworks from Smalltalk-80/Windows 3.0. The package includes Parc Place Systems' Objectworks integrated development environment and an extensive class library, which features more that 300 reusable software modules according to a company spokeswoman. ... "

Objectworks for Smalltalk-80/Windows 3.0 will cost $595.

Widgets/V 286

Acumen Software
851 Lytton
Palo Alto, CA 94301
415 328-3816

Widgets/V 286 is a user interface development package for Digitalk Inc.'s Smalltalk/V 286. Smalltalk provides an object-oriented language for rapid software development and prototyping, but lacks powerful tools for user interface design. Widgets/V 286 fills this gap with an interactive interface editor for point and click development and a user interface object library for building block reusability. The interface editor extends the interactive, exploratory nature of Smalltalk development to the design of user interfaces. Using this tool, an interface is built by moving, sizing and editing the widgets directly on screen. With the editor's run mode, developers can switch back and forth between designing and testing applications.

The object library covers a broad spectrum of visual objects used in graphical user interfaces. These objects include various window styles, hierarchical popup and pulldown menus, listboxes, a full featured text editor, scrolling panes, radio buttons, palettes of forms and others. The existing widgets can be subclassed as desired and a set of intrinsics is provided for creating customized widgets with support for input handling and graphics.

Widgets is being offered at an introductory price of $99 plus shipping and handling and will begin shipping on September 1, 1990.

ACTOR–OS/2

The Whitewater Group
Evanston, Ill.
708 328-3800

Platform News ("Systems/3X and AS World," June 1990) reports that "The Whitewater Group ... has introduced an OS/2-based version of the ACTOR language that utilize Presentation Manager. Earlier versions of ACTOR were DOS-based and used the Windows environment. ... ACTOR is similar to Smalltalk in that it allows only object-oriented statements and no procedural statements."

PRODUCT NEWS

Objectworks\C++ Release 2
ObjectKit\C++

ParcPlace Systems
1550 Plymouth Street
Mountain View, CA 94043
415 691-6700

Objectworks\C++ Release 2 is an integrated development system designed to help C++ programmers exploit the benefits of object-oriented technology. The product enables programmers to create and reuse C++ code to build robust applications efficiently. ObjectKit\C++ is a collection of reusable class libraries including AT&T Standard Library Extension.

Significant new features of Objectworks\C++ enhance the openness and integration of the environment, permitting users to work with the product in conjunction with traditional UNIX tools. In addition, the product has performance advancements to support large system development and incorporates the AT&T C++ Language System Release 2.1. The extensive C++ class libraries provide the benefit of code reuse.

Objectworks\C++ includes the following new features:

Open Environment: allows users to select their preferred C preprocessor, C compiler and linker to customize the development environment. For example, those who prefer the GNU C compiler can use it instead of the default SUN C compiler. In addition, developers can use Objectworks\C++ in conjunction with source code control systems, profilers and debuggers developed by third party vendors.

Integrated Environment: Because Objectworks\C++ is integrated into a single systems, developers save time and are guaranteed accuracy. The product's source level process inspector allows programmers to step into and debug C as well as C++, providing a migration path from C to C++. The product also provides a consistent editor and look-and-feel.

Support for Large Systems: the performance has increased significantly for faster compilation of larger programs. The development environment now offers faster import and turnaround times and the internal object-oriented database has been tuned, reducing the memory requirements. Because the product can be used in conjunction with third party source code control systems, the development environment facilitates team programming projects.

Class Libraries: ObjectKit\C++ is a collection 15 general-purpose C++ libraries, including AT&T's Standard Library and Standard Library Extension. ObjectKit\C++ also bundles the National Institute of Health C++ libraries, which are translations of selected ParcPlace Smalltalk class libraries and InterViews, an X-windows graphical user interface developed at Stanford University. Characteristic of the object-oriented approach to programming, these libraries facilitate greater programmer productivity and higher quality program development. Basic Objectworks\C++ components include:

Inheritance Browser	Error Browser
Call Relationship Browser	C++ Translator using AT&T Language System 2.1
Program Structure Browser	C++ Source Level Debugger

Objectworks\C++ will be available for the Sun-3 and SPARCstation platforms in September 1990. The memory requirement is 12MB and the price is $3000.

ObjectKit\C++ is priced at $500 and will be available in October 1990.

only then, is the task of implementing the system begun.

The goal of the analysis phase is to answer not *"How is the project going to be done?,"* but *"What is going to be done?"* The result of this phase is a set of documents that define the information model of the system. This traditionally embodies entity–relationship diagrams (ERD), functional hierarchies, and data flow diagrams. This method divides the information model into a data model and a functional model.

What distinguishes the object-oriented approach is that the two (data and function) are not separated. The document used to describe the information model is the object description sheet (ODS). As I will show, this small step, the reintegration (of what should have never been divided in the first place) of the data and functional model, is easily taken and produces a tremendous improvement in how useful the analysis phase is to designers, programmers, and even users.

The process of creating the information model begins with defining the words an end user uses daily. Requiring both users and programmers to define and agree on a common set of nouns (used to describe the project) assures us that the project will meet the user's needs. This is the primary reason behind defining the vocabulary of the project.

FINDING THE OBJECTS

However, of equal interest to us is that the objects in the system become immediately apparent to the O-O designer: each word in the lexicon will become an object. Why, you ask, if it is so simple, does everyone always ask *"How do I find the objects?"* The reason is that people begin with language instead of design, or, rather, people begin with programming language instead of the users' language.

As an example, let's examine one of the prime sources for finding the objects in a system: data entry forms. These forms, probably created by the users, are employed regularly and represent the most basic information entered into the system. Furthermore, to facilitate the use of the forms, related information will be grouped together, probably into discrete sections.

Each of these forms will become an object. The simple forms will become simple objects, whose attributes are basic data types. The complex forms with many sections will invariably become complex objects, whose attributes are other objects, based on the sections of the form. In addition, every task that uses the form will become a method. If the task relies only on the information in the form, the method will belong to the object that represents the form. If the task revolves around another object, it will become a method of the other object, which takes as a parameter the object representing the form.

> ***"The first word that needs to be redefined is the word "OOPS." Certainly, no one can take seriously a technology that sounds like a mistake."***

When introducing O-O design, it is essential to realize that, although the words the object-oriented community uses are different, they refer to (basically) the same concepts programmers and designers have been using for years. For instance, "entities" in ERDs become "objects" in object–relationship diagrams (ORD).

If all we do is change the word "entity" to "object," we will have completed 70% of the move to O-O design. The interview process, document analysis, and feedback sessions between analyst and user remain the same. Even the use of a CASE tool needn't change, although CASE tools built around a repository, like Oracle's Case*Designer, are easier to use because they can be modified to hold the O-O information. Each entity gets defined as do all of its attributes. In defining the types of attributes, most CASE tools require that the domain of the attribute be specified. This is just a fancy way of giving the attributes data types. Some of the domains will themselves become objects. The organization of entities as sub and super types is also carried out. This is just another way of saying to an O-O designer that some objects[1] inherit from other objects. Thus, after the ORD is done, the objects, their attributes, and the class hierarchy specifying inheritance relationships will be defined.

Thus far, the analysts have been doing what they've always been doing, with the addition of talking a new language. As a way of introducing object orientation, one of the easiest things for new O-O designers to grasp is that relational modeling, specifically in terms of ERDs, works equally well for objects. Furthermore, because the end result of the model is not solely database design, but, rather, includes classes inside the program, the designers are urged to carry normalization to its extreme. Each entity that would have

been too small for a database gains significance as an object that stands alone.

THE DATA FLOW DIAGRAM

Once the ORD is completed (and thus, all the nouns of the system will have been defined and agreed to by the users), the remaining tasks serve to confirm the choice of objects and to expand on their behavior.

The functional decomposition, while having no direct corollary to O-O terminology, is used to define the business functions carried out by the system. Each business function is defined by an English description using the agreed-upon vocabulary of objects. The functional decomposition will show which objects are needed by function and what functions the individual objects are going to carry out. *Voilà!* We have now defined the methods to be implemented in the system.

In addition to defining the methods the objects will have to support, the functional hierarchy will show whether the choice of objects was optimal. By seeing what kind of information each object contributes to each function, we can apply the maximum bandwidth rule[2] to each object to see if we have correctly encapsulated data and functions.

A data flow diagram (DFD) is used to show how a business function is completed in relation to the information flow into and out of the function. In O-O terms it shows how information flows from one object to another to complete a task. The DFD consists of datastores and dataflows. Datastore is another word for object, or on detailed DFDs it can be the attributes of an object. Sometimes these datastores do not represent intrinsic objects but derived objects.

The DFD is particularly good at showing which pieces of information and which methods should be encapsulated together. By looking at the number of dataflows and the information on each dataflow into a business function, we can easily see whether the objects chosen pass the bandwidth and throughput tests.

By renaming the terminology of CASE, relational database design, and traditional programming, we have accomplished 90% of our goal in introducing O-O analysis and design. The designers will soon be using the new terminology and with it the slight change of priorities required for O-O design. The new priority is to think of an entity as an object with all of its newly-attached behavior.

The main deliverable associated with object orientation is the object description sheet. Each sheet completely specifies an object. It is produced as a result of integrating the data and functional models. For every class of objects, its superclass, textual description, attributes, methods, relationships to other objects, and contractual relationships[3] with other objects is listed. The ODS is created in the analysis phase, with a draft of its attributes, methods, and a description. As the project moves into design, the rest of the ODS is completed. A finished ODS is a complete specification a programmer can use to implement the object in an O-O language. By encapsulating the attributes and methods, and the requirements an object has of other objects, a programmer may safely code the object without causing havoc at the integration phase of the project.

THE INTEGRATION PHASE

If the implementation environment is not object-oriented, the team must only adhere to a set of rules to ensure that integration will be easy. The main drawbacks in the object-orientedless environment is that inheritance will not be available. All other features of O-O programming languages (encapsulation, message passing, and polymorphism) can be realized through a set of programmer guidelines. The two main guidelines specify the use of header files and naming procedures.

If the project is heavily DBMS-dependent, the methods attached to the objects (some of which will be database tables) will not be integrated with the data. Thus, additional guidelines to guarantee data integrity will have to be specified. Since the O-O community has yet to solve the OODBMS problem, this is not a major drawback.

SUMMARY

The object description sheets are used in the full lifecycle of the project — they aid in specifying the requirements of the users; they provide the programmer with complete specifications; they are used to write the technical documentation of the project; and finally, because the object names are isomorphic to the words users use to describe their work, the ODS can also be used to write the user documentation.

If such an approach is taken, I would bet the next project undertaken will rely not only on O-O design, but also on O-O languages. As they say, success breeds success.

The source of the approach outlined here is object orientation itself. There are two basic conceptual objects: design and programming. The design object requires input from users and analysts and produces specifications. The programming object requires specifications and produces a system.

However, there are two subclasses of the programming object: O-O programming and traditional programming. Furthermore, O-O programming is a subclass even more specialized then traditional programming. It requires not just specifications but a specification for each object to be programmed. Thus, the O-O programming object can't run unless it is supplied with additional information. However, the traditional programming object will run on both limited and O-O specifications. It runs faster and more efficiently with O-O specifications.

In the end, there is a reason for the jargon: it refocuses designers and programmers alike to concentrate on the objects in the system. By concentrating on the objects in the system, the ability to deliver what the user requires is enhanced; and that's what it is all about. ■

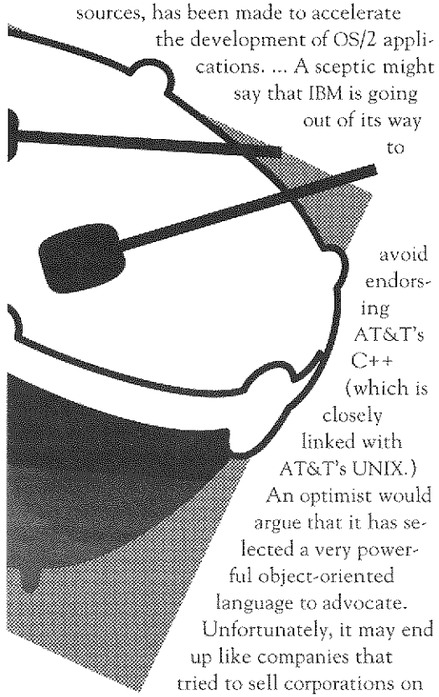
Footnotes

1. In this article, I have referred exclusively to objects and entities. In the relational de-

(Belmont, Calif.) are suppliers of advanced, massively parallel computing systems and software.

Source: **Supercomputing Review**, May 1990.

At the same time that Borland released C++, IBM has begun encouraging OS/2 Presentation Manager developers to use Smalltalk/V to develop new applications. It is even rumored that IBM might make Smalltalk/V part of its SAA standard. The move, according to IBM sources, has been made to accelerate the development of OS/2 applications. ... A sceptic might say that IBM is going out of its way to avoid endorsing AT&T's C++ (which is closely linked with AT&T's UNIX.) An optimist would argue that it has selected a very powerful object-oriented language to advocate. Unfortunately, it may end up like companies that tried to sell corporations on using Lisp for expert systems development. Sure Lisp is powerful and flexible, but...



Source: "IBM Adopts Smalltalk/V for OS/2 Development," **Expert Systems Strategies**, Vol. 6, No. 5, 1990.

The Object Management Group (OMG) recently announced that DEC has become a corporate member and will serve on the OMG's board of directors. ... According to Kastner [Peter, a vice president at the Aberdeen Group, a Boston-based market research and consulting firm], DEC's Compound Document Architecture (CDA) could be one impetus for the company's involvement in the OMG. ... "DEC's entire CDA is based on the concept of documents with live objects from databases, spreadsheets, CAD drawings or other sources," he said.

Source: "DEC Joins OMG, Touts Object-based Applications as Standard," **Digital Review**, Lois Slavin, June 18, 1990.

"Object-oriented programming is a big thing, it's a big bang, it's trendy and it's neat. It really offers productivity benefits, but everyone's afraid that if they require so much to learn a new language, nobody will do it. So, it's a common thing today to graft these object-oriented constructs onto procedural languages. That's always kind of a hack, but it can be done neatly and elegantly. Apple has hired the inventor of Pascal, Niklaus Wirth, to come in to help it with Pascal. And that gave us Object Pascal, which is pretty nice. It's clean, it's simple, there are only a very few expansions, but they're powerful."

Source: "A Snapshot Inside Apple From David Ramsey," **Byte**, Jean Mickelson, May 1990.

ANTI-HYPE

In fact, the term has become so popular it has begun losing its meaning, with not only object-oriented languages, but object-oriented databases cropping up all over. Like "fourth generation" and "knowledge-based", "object-oriented" has become a buzzword. ... This doesn't mean there is no reality behind the hype. Particularly in the microcomputer field, several commercial language compilers use object-oriented concepts. ... But some question the concept itself. Paul Bassett, vice president of research at Toronto-based Netron, Inc., calls object-oriented programming "a lot of old ideas dressed up in new garments. It's the subroutine library with new calling procedures." ... Bassett favors making code reusable, to reduce programming time and cut down on errors. But he says the object-oriented concept of inheritance, which he defines as "same as plus", is too unwieldy.

Source: "Languages, Methods, All Part of a Cobol Evolution," **Computing Canada**, Grant Buckler, June 21, 1990.

"The problems we are attacking now are so complex that they are beyond the scope of traditional languages," says Bjarne Stroustrup, author of the C++ language. ... Object-oriented programming sparks the interest of programmers because it has the potential to resolve these development issues. "In the old days, it took a lot of effort to do very little," Stroustrup recalls. "And what you ended up with required tedious debugging. The object-oriented paradigm is taking the tedium out of programming."

Source: "Programming by Components," **CommUNIXations**, Laurie Miller, May 1990.

STANDARDS

The Object Management Group is seeking industry comment on its reference Model, Abstract Object Model, and Technical Objectives, which will be part of the organization's Standards Manual. The Reference Model offers design guidelines; the Object Model is a dictionary of terms and relationships for object-oriented development; and Technical Objectives lists characteristics of technologies under consideration. Copies of the proposals are available by calling 508-820-4300, or sending a fax to 508-829-4303.

Source: "Comments Requested for Proposed Standards," **Infoworld**, June 18, 1990.

ponent based; component classes may be specialized through inheritance, and component instances may be structurally composed into part hierarchies and their ports interconnected to accomplish behavioral interaction.

Source: "Simple/Care concurrent-application analyzer," **IEEE Software,** Nakul Saraiya, May 1990, pg. 27.

In his April 17 Tech Files column [Joel West] makes three points that cry out for clarification.

First he takes a swipe at Apple, blaming the company for what he sees as a silly attempt to change the terminology from object-oriented programming (OOP), to object programming. Actually, I think credit for this cleaner description should go to Scott Knaster, an Apple engineer and writer, who suggested at a meeting of the Software Entrepreneurs Forum a year ago that the word "oriented" didn't add anything to the description. I completely agree. ... Finally West makes the statement that "not one leading application has been written using Apple's OOPS tools." Aside from the fact that the same thing could be said of every other platform, I wonder if he's seen such wonderful programs as GeoQuery, Photoshop, Stratavision 3D, Micro TV and Read-It! All are written in MacApp.

Apple has made a clear and strong commitment to OOP (oops, I mean OP) technology. The company may not have done as much to promote and support it as a lot of us would like, but Apple is a lot further along than it got credit for.

Source: "Letters: Shafer Credits Apple with Strong OOP Commitment," **MacWeek,** Dan Shafer, June 19, 1990.

The imminent release of second-generation object-oriented database management systems has created a stir among computer-aided software engineering tool suppliers and database suppliers. ... The companies preparing to unleash OODBMS products in the next quarter are Object Design, Burlington Mass., which will ship ObjectStore, and Versant Object Technology, Menlo Park, Calif., which will release Object-Base. ... "Everyone agrees that traditional relational database systems do great on business data processing, and lay an egg if you ask them to do anything else," said Michael Stonebraker, professor of computer science at the University of California at Berkeley, and a founder and member of the board of directors at Ingres Corporation, an Alameda, California-based relational DBMS vendor. "If you want them to store documents or CAD-related information, they just don't do very well. That's universally agreed. The only question is what to do about it."

Source: "Objects are Taking Shape in Flat Relational World," **Software Magazine**, Ali Hazzah, June 1990, pp. 32-42.

LANGUAGES AND TOOLS

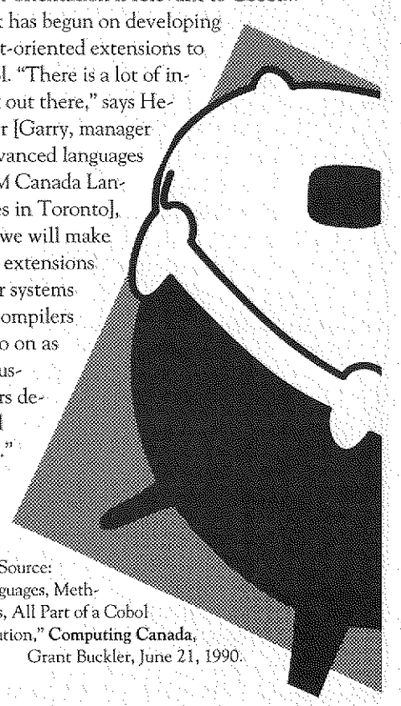
The 12.5 MIPS Sparcstation now supports the Simula object-oriented programming language. Standard implementations include libraries for common problems in mathematical functions, random number generators, text manipulation, standardized file system interfaces, linked lists, and discrete event simulation.

Source: "Educate, develop programs with Simula," **IEEE Software,** June 1990.

Interactive Development Environments (IDE) and Interactive Software Engineering (ISE) are interacting on an upcoming new product release. The companies plan to create an interface between IDE's Software Through Pictures CASE environment and ISE's object-oriented programming language, Eiffel, to allow integration of the two products. ... "It's one thing to represent an object-oriented design, but you still need and object-oriented language to execute that design," Niehaus [Ed, spokesman for IDE] said. ... Contact IDE: 415-543-0900.

Source: "OOP-CASE Interface," **UNIX Today**, June 11, 1990.

...Cobol's proponents are not so sure of their ground that they ignore new developments. The committees that watch over Cobol standards have agreed that object-orientation is relevant to Cobol... Work has begun on developing object-oriented extensions to Cobol. "There is a lot of interest out there," says Helander [Garry, manager of advanced languages at IBM Canada Languages in Toronto], "and we will make these extensions to our systems and compilers and so on as our customers demand them."



Source: "Languages, Methods, All Part of a Cobol Evolution," **Computing Canada**, Grant Buckler, June 21, 1990.

OOT IN LARGE COMPANIES

MasPar Computers Corporation and MIMD Systems are joining forces on a NASA-funded database study on applying parallel computers to object-oriented data management. Funded by NASA's Jet Propulsion Laboratory (JPL) in Pasadena, Calif., the project will adopt MIMD Systems Arachnid object-oriented data manager to MasPar's SIMD architecture. ... Both MasPar Computer Corporation (Sunnyvale, Calif.) and MIMD Systems

sign approach, entities represent both the object-oriented concept of object and class. O-O mavens will thus please excuse this ambiguity. It will become clear to any user what a class is after using the O-O design techniques.

2. There are two rules that should be applied to test whether objects have been correctly encapsulated: the bandwidth rule and the throughput rule. The bandwidth rule states that the information exchanged by two objects for one operation should be either a single object or a very small set of intrinsic data types. This is a qualitative rule used to make sure methods do not require so much information that they violate encapsulation. The throughput rule states that to complete a task a single object should be sent a minimal number of methods. This is a quantitative test that will show whether the boundaries between the objects were chosen correctly.
3. The contractual relationship between two objects is the specification of what methods the receiving object expects the parameter object to understand/implement.

Seth Copen Goldstein, president and founder of Complete Computer Corporation, graduated from Bronx High School of Science and Princeton University. He is the author of Complete C, the object-oriented toolkit for C programmers. Goldstein currently consults on full life-cycle software methodology and object-oriented design.He can be reached at Complete Computer Corporation, 111 West 57th Street, New York, NY 10019, telephone (212)582-2635 or Fax 212-397-1580.

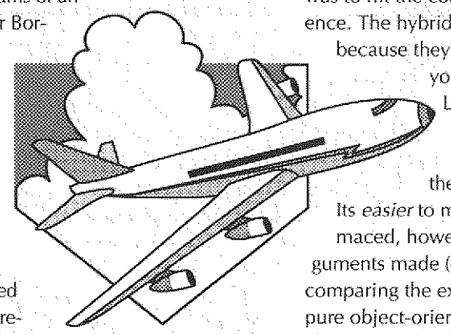
≡≡REVIEW

Objects on tour

Marie A. Lenzi

"World tour" conjures up images of The Rolling Stones, Aerosmith, or Madonna pillaging and plundering their way from London to Tokyo. It was with this image in mind that I attended the New York performance of the Borland OOP World Tour.

Conflict arises when expectations and reality don't mesh. Early into this day of introductory OOPS, I experienced this conflict. The overall appearance was indeed professional, lightly theatrical, but never intimidating or self-serving. However, my worst dreams of an eight-hour Borland commercial were never realized. In fact, there was a concerted effort to prevent the day from becoming a detergent ad.



The materials covered were a general overview of the technology, management, both technical and organizational, analysis and design concepts/issues, C++, and Object Pascal. The technology was covered at an introductory level. For the most part, the attendees had done reading in objects but did not have much practical experience. Over the course of the day, I asked people how they liked the conference. Early on, most were unable to articulate their feelings; but later in the day, as the parts began to come together, light bulbs started to turn on. During the day, few people left and virtually nobody skipped sessions to loiter around the coffee and soda. In fact, people returned promptly to the sessions.

The three presenters know their jobs and their materials well. David Intersimone of Borland covered the overview and management and design topics, Bruce Eckel of Revolution2 did the C++, and Borland's Zack Urlocker (one of the clearest presenters of the technology) covered the Object Pascal.

The main thrust of the presentation was to hit the comfort zone of the audience. The hybrid languages are *easier* because they feel so much like what you are used to using. Look, see C++, Object Pascal? They more *easily* live side by side with the existing body of code. Its *easier* to migrate, etc., etc. I grimaced, however, at some of the arguments made (or omitted) when comparing the extended languages to the pure object-oriented implementations. There are enough arguments on both sides to present the facts and let the buyers decide for themselves. But, even though they hit the "ease" concept a little heavy, they didn't trivialize. Facts and reality were clearly laid out.

At the end of a review, there is supposed to be an "although," but I really can't come up with one. I feel Borland has done a good job of presenting introductory OOPS simply, professionally, and competently, with little corporate hype. I think the attendees got value for their dollar. And, for once, the fact that my expectations and reality didn't mesh created no conflict. ▮▮

ANALYSIS & DESIGN

Design principles for modular product development

Douglas W. Bennett

We hear regular calls for significant changes in the way software products are developed. Mitch Kapor put it nicely at a recent PC Forum when he called for "radical advances" in the way software products are designed and developed. Another word for *advances* is *changes*.

What is the nature of those changes, and what will bring them about? Much of the published discussions about change involves tools: O-O programming languages, CASE tools, etc.; but this emphasis on technology tools does not address the root problems. I believe Kapor is correct in placing the emphasis on the way software is **designed** and **developed**.

In this article, some design priorities for software products are proposed. The motivation for these suggestions is to deliver a quality product — one that meets the needs of the users, developers, and sellers of the product. These priorities lead to a different way of designing software products than is commonly used. I will describe some specific techniques that support those priorities and outline a process that incorporates those techniques.

PROBLEMS AND PRACTICES

The current problems with software products have been well documented. Many of those difficulties can be traced to several development practices common today, including:

- The use of deep decomposition of data and functionality as the primary means of both analyzing and structuring an application;
- The lack of emphasis on communication interfaces at all levels of a product;
- The lack of explicit evaluation of behavior allocation decisions, in spite of the best efforts of Parnas, Page-Jones and others;
- The inclusion of transient entities, functions, and data objects in the structure of the application;
- A lack of design notation showing a product consisting of an assemblage of tangible components.

The first item encourages the building of systems from pieces at the bottom of the decomposition and discourages any meaningful layering. In functional programming, the pieces at the bottom are single purpose functions and dumb data. In OOP, the pieces at the bottom are instances of a class in the implementation language. In both cases, building a large system from small units makes the system appear very complex.

Such complex systems also appear to be monolithic and so discourage examining the communication interfaces.

The last two items on the list help ensure software will be fuzzy — always changing, and, therefore, not amenable to conventional engineering design practices.

PRODUCT ATTRIBUTES

The end result of the radical changes to the software development process Kapor called for, I believe, will be software products with the following characteristics:

- Software architectures will look like electronic component architectures, i.e., the components of the architecture will remain fixed while the product runs;
- Products will consist of several layers, each with their own architectures and kinds of components;
- Communication protocols will be an explicit part of the software architecture at each level; protocols will be carefully designed for robustness and stability. A market for individual software components will exist only when those protocols are standardized across enough systems to support a business;
- Many of the components at each level will be standalone, atomic units that can be picked up and used (reused) in other applications.

Some of these product attributes sound like benefit claims being made for object-oriented programming languages. O-O programming, by itself, is just as susceptible to the problems and consequences mentioned above as conventional programming. I believe the desired attributes will appear in products because they are explicitly designed in, not because of the particular implementation language or CASE tools used. If that is the case, what kind of design priorities and practices will yield the desired results?

DESIGN PRINCIPLES

For this discussion, consider the process of designing one layer of a software product to consist of developing an external model,

PUBLICITY CORNER

APPLICATIONS

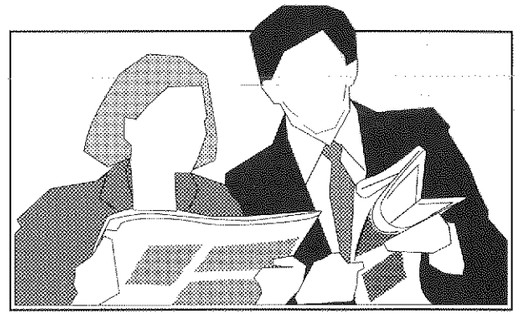
...at some undefined date in the next few years, Borland will ship an object-oriented database management system (referred to at the recent Paradox user group conference) called Chameleon.

Actually , that last sentence should probably have read, "a database called Chameleon that Borland refers to as object-oriented." ... What Borland plans to deliver, and what it seems we're likely to get from most of today's major DBMS vendors, is really an object-oriented interface builder with an installable file system (in Chameleon's case, you'll be able to chose Paradox, Dbase, or SQL files). In other words, the tools to manipulate the graphical user interfaces and application logic will be object-oriented, while the data access mechanism will remain relational...

Source: "The Brink of War," **DBMS**, Kevin Strehlo, July 1990, pp. 8–10.

"The extended relational/restricted-object-oriented database will lead the marketplace," says Ladd [Ivor, VP and co-founder of Empress Software Inc. in Greenbelt, MD] ... Both Francois Bancilhon, a researcher at Altair, a consortium based in France, and charged with producing an object-oriented DBMS, and David Choy, a researcher at IBM's Database Technology Institute within the Almaden Research Laboratories, felt compelled to deflate the hype surrounding trendy adjectives for databases.

"Everybody and his brother is now object-oriented without knowing what it is," stated Bancilhon matter-of-factly. "A multimedia database is a DBMS designed to support a variety of multimedia applications as well as traditional ones, nothing more," said Choy. ... "The next generation of database systems will be object-oriented, starting in niche markets and addressing the conventional markets by the year 2000," said Bancilhon. "The technology to build the system is about ready." ... "Object-oriented databases don't tend to worry about how can I modify something when someone else has it," said Ladd. "Object-oriented is poor for multi-user and yet, to be commercially viable, you have to be multi-user."



Source: "The Story of 'O'," **Personal Computing**, John Blackford, June 1990.

Source: "Object-Oriented is Next Wave for Databases," **Computing Canada**, Doug Powell, June 21, 1990.

OOP products fall into four basic categories: operating systems, programming languages, user interfaces and databases. This breakdown is reflected in the products that were introduced last year, suggesting that the major software vendors are going after the OOP market on all four fronts. The most hotly contested battles, however, will be fought in the database management system (DBMS) arena.

Source: "Software Stays at the Cutting Edge," **Datamation**, Harvey L. Poppel, June 15, 1990, pp. 194–195.

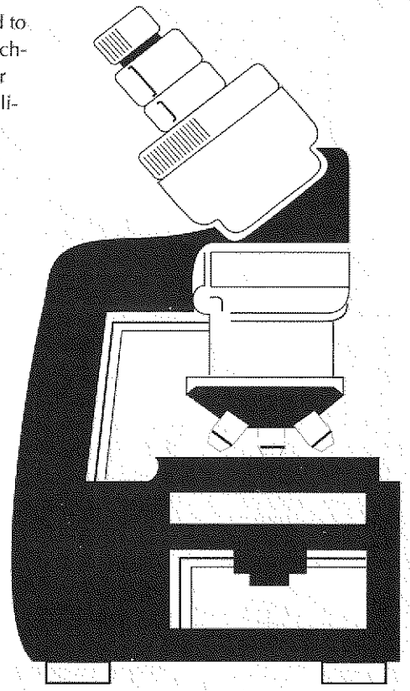
While software growth in the United States has slowed to less that 20% per year, object-oriented software is increasing at 110% annually—although this rate is based on limited numbers so far. Before object-orientation reaches the mainstream, developers and users must face a day of reckoning that will require a fundamental shift in software design—because software reaching full object potential is best built from the ground up, a fact that practically guarantees a turbulent transition.

Still the myth of programmerless programming endures. And why not? The idea is compelling. It's exciting to think that the full problem-solving power of the personal computer might become available to more than the proud few who have mastered the intricacies of interrupts and pointers. ... Lately I've been preoccupied with a vision of a programmerless programming environment that just may be within the realm of feasibility within the next few years. I say "may be" because I haven't thought through all the technical problems yet and I don't know if anyone is working on such a system. The inspiration comes from my research into code reusability and my growing appreciation for object-oriented programming.

Source: "Object Soup," **Computer Language**, J.D. Hildebrand, June 1990.

Stanford University's Simple/Care tool lets you execute applications developed in a concurrent object-based language on multiprocessor system models that have been specified for a general purpose simulation system. ... At the heart of the tool is the Simple object-oriented discrete event simulator, which lets you specify multiprocessor designs and their instrumentation. The design interface is com-

Modeling (Entity–Relationship modeling), you can add a measure of sophistication by building an information model reflecting real world objects and the relationships between them. The doctor believes that this is critical to the success of any but the most trivial O-O efforts. Information modeling is a mature body of knowledge and is clearly better than unaided observation of the system environment.



You will need to use a different technique to discover the essential application classes. The most important essential application classes participate in event processing. Event modeling will help you uncover those.

Recall that an event is an external event if it happens in the system's environment and if the system is responsible for making a response. A Patient Monitoring System event might be "medical staff enables patient monitoring" (the system must respond by starting to collect data on the patient) or "patient temperature exceeds threshold" (the system responds by generating an alarm to the medical staff). "Sensor interrupts" is an event, but at a different level of abstraction. It is addressed as part of foundation class design.

The system will need event stimulus recognizers to discover the events. They wait for events, pounce on them when they happen and hand them off to the event manager or administrator whose job it is to respond according to requirement. Each manager and administrator uses the services of the essential business classes by delegation. The event handling classes are the most application specific of the classes we've explored so far.

They, along with the essential business classes and foundation classes, are the minimum raw material the Doctor explores when he considers the potential object classes in a system.

So, now back to your original question: how do we find object classes? The doctor finds the essential classes first. He is rather more concerned with what needs to be done rather than how to do it.

Discovering and modeling the essential business classes and essential application classes might be done serially or in parallel. It depends on the size of your problem and the size of your staff. As your object library matures you will begin to reuse the essential business classes by introducing new event handling machinery and therefore new functionality. In generally, the domain of essential business classes is more finite than what can be done with them.

After dealing with the nature of the problem, the Doctor is ready to derive a solution. Having made decisions about the platform, the language and the data management, the communications protocols, and the environment, he identifies the foundation classes that implement the services the essential classes require. Some of the foundation classes may already exist. He will need to build others.

These design activities will be idiosyncratic to the environment. The doctor's advice to an engineer working with C++ is very different from the advice he offers to an engineer using Eiffel. ▮▮

The Doctor has been a consulting methodologist for over a decade. He invites you to submit your questions on object-oriented technology to:

Dear Doctor
1202 108th Avenue East
Puyallup, WA 98372

an internal model, and an implementation architecture.

Design priorities

I propose that any design process that supports the product attributes described earlier should include the following:

- The process should proceed in layers, from requirements through architecture specification for each layer. Decomposition during analysis, or modeling, is only one or two layers. The unit of decomposition is the implementation component. The process of deciding how to decompose one component is the entire design process for the next layer.
- The behavior of all components, in both the modeling and architecture phases, is described in terms of the protocol, or messages, used to invoke the behavior.
- Identification of the required stable architectural components is a separate process from analysis and modeling. Modeling is concerned with accounting for all the control, processing, and data needed to meet the user's requirements. The architecture phase is concerned with identifying implementation units that will make the product maintainable and extensible — in other words, the architecture is concerned with meeting business requirements.
- Each time a new set of components is identified during modeling and architecture in each layer, the behavior allocation implied by the protocol assigned to each component should be explicitly evaluated to ensure strong modularity has been achieved.

Comparison with conventional design priorities

For purposes of discussion, conventional design refers to the various flavors of structured analysis and design consisting of functional requirements, data modeling, and behavior modeling with hierarchical data (and control) diagrams.

The major difference between the principles proposed here and those of conventional software design and analysis is in the decomposition of the system.

Conventional analysis uses decomposition repeatedly until function or data primitive is reached; the alternate proposed here specifies one level of decomposition in modeling is followed by a process leading to implementation (architectural) components, which are then subject to the next layer of analysis, modeling, and implementation unit design. To put it another another way, analysis does not penetrate more than one of the opaque layers at a time.

> ***... any design process, by itself, does not confer any certainty or correctness to products.***

By expressing all behavior in terms of message requests, the "dumb data" of conventional analysis is eliminated: all components have behavior as well as data and state — i.e., everything is an object.

Communication protocol and interfaces within software subsystems should receive the same attention as is given to hardware interfaces in physically distributed systems. This is in contrast to the common practice of considering interface only between hardware components.

Comparison with OOD principles

Object-oriented design approaches, now growing rapidly in number, seem to have several common threads. One is an emphasis on modeling the real world, with the implication that all objects in a system should be real world things.

Another is that all objects in a product can be directly implemented as instances of classes or types in the chosen programming language. My experience with the process discussed here has shown that, while these two principles of OOP are useful and necessary, they are too restrictive to deliver the promised benefits to real products.

The use of opaque layers in a design implies that there will be objects at each layer and that the objects will have different implementations at each layer. For one implementation layer, instances of a programming language class may be used. At a higher level, a Unix task or a node on a network may be the appropriate implementation for objects.

If an architecture of stable objects is the goal, physical modeling is not sufficient. The process of identifying and allocating behavior to those architectural components is different, and should be separate, from the physical modeling process. Objects from the application domain are a good start, but there are several types of product domain objects needed to provide a stable and robust product architecture.

EXISTING TECHNIQUES

If we define the process of designing one layer of a software product as consist-

ing of developing an external model, an internal model, and an implementation architecture, we can ask if there are any design techniques that support these kinds of priorities. There are several; they are not mainstream techniques, but neither are they new or untried. I have used three individual techniques to assemble a design process that supports the priorities proposed here. The techniques and their sources are as follows:

1. The CORE requirement modeling method from England, with some extensions, serves well for the external model. This technique was developed by British Aerospace and SD Scicon in the late '70s. Further development of the method is being carried out by SD Scicon. This technique bridges the gap between conventional, text-based functional requirements and the kind of modeling needed to describe the internal working of a software product.
2. The Objectory method, published by Ivar Jacobson, provides a very layered, communication centered approach for developing the internal model. This approach is based on the design techniques used by Ericsson Telephone in Sweden for telecommunication systems for the last 20 years.
3. A notation and method for describing software architecture in terms of fixed components that communicate with each other is provided by the Mascot method. This technique has been used in the British military for design of embedded systems since the early '70s.

> ***"... the layering proposed here will prove to be the path that software development will actually take."***

A DESIGN PROCESS

I will briefly describe a design process that combines the priorities and techniques mentioned here. Each layer has a modeling phase and an architecture phase. The output of the architecture phase specifies the implementation for that layer. Getting from requirements to architecture involves building models of the target system and its environment as it appears from the outside, the external model; a static (data or object) model of the system internals; a dynamic or behavior model of how the system will respond to each of the external events or commands described in the external model; and an architecture model in which the fixed, implementation units that will comprise the system are developed.

Each model listed above can be constructed in three steps: identify components (objects), allocate behavior and the protocol to invoke the behavior, and review the components and the allocated behavior for completeness and modularity.

In the external model, the components are both the target system and the physical entities, including people, that interact with the system. The behavior to be allocated is described by the functional requirements. The static model components are all the physical and data objects the system must handle. We are not explicitly concerned with behavior in the static model, but, instead, with relationships between the static objects.

In the behavior model, the components being identified are active objects that manipulate and transform the passive objects of the static model: process and control components. The behavior being allocated is that functionality needed to deliver the required event responses.

This behavior, roughly the functions of conventional data flow analysis, is allocated to whatever component, active or passive, seems appropriate. After reviewing both the identity of each component and the behavior allocated to it for modularity, the result is a set of objects that completely accounts for all the required system behavior. If the review was done well, they are strong, atomic components with well defined interfaces.

These components are not, however, a sufficient basis for specifying a product architecture, because many of them are transient, i.e., created, destroyed, and moved around during operation of the system. The architecture model addresses the need for stable components.

The architecture model identifies components that exist in a fixed position as long as the application is executing. An example of a minimal set of architectural components would include one component to handle I/O with each person or device that interacts with the system, at least one component to house the services the application offers to its users, and at least one to house the model manipulated by the services.

To qualify as an application service, the object must have multiple dependencies in the model or the I/O components. The behavior allocation step in the architecture model is made easier by the fact that all the required behavior has already

sents Mr. Booch's methodology for designing complex object-oriented systems. Here's where my opinion begins to vary widely. It's also where I'll attempt a novel approach to book review, one that never discusses the content of the nearly 600-page book (see table).

As they say, readers are left to form their own conclusions.

The next section, *Pragmatics*, is considerably better. It offers useful suggestions on integrating the object-oriented techniques into the project lifecycle, the organizational structure, and the corporate methodology, if any.

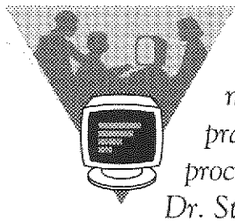
THE THIRD BOOK: APPLICATIONS

If you've already been introduced to the concepts of object orientation, and especially if you've tried a couple of them, the applications section is the best part of this book. In it, Mr. Booch presents a wide variety of systems implemented in an equally wide variety of languages. Are you a real-time systems expert? The home heating controller in Smalltalk or the railroad traffic manager in Ada will tell you far more about the differences in an object-oriented design than the lamentably short design section. Are you a commercial developer? The decision support system, implemented in C++, is for you. An interactive graphics system, implemented in object Pascal and utilizing the MacApp application framework, and a cryptanalysis system in Lisp round out this section.

A word of warning. If you are not familliar with a fair number of these languages and are not already comfortable with object orientation, it will be difficult to separate elements of these designs into O-O features and implementation language artifacts. ▮▮

Bob Howard is the Director of Technical Services at The Whitewater Group and is the co-author of the Object Graphics class library. He can be reached at 708-328-3800.

≡≡ DEAR DOCTOR



A forum on object-oriented methods, practices, and procedures. Dr. Steven Weiss attending.

Dear Doctor:

We are a newly object-oriented shop. We are designing and building our first major system. One of the most difficult problems we have is finding object classes. Can you give us any advice about where to look for them?

Yours is one of the questions the Doctor is most frequently asked. Objects can be organized into several categories or taxonomies. One such taxonomy arranges object classes according to their degree of specialization.

Essential application classes are the most highly specialized. They're typically unique to the product you are building. Essential business classes are less specialized. They're unique to the business you are in, but are common to more than one of that businesses' applications. Foundation classes (also called kernel classes or core classes) aren't terribly specialized at all. They are used across many applications and businesses.

Let's say that you're building a Patient Monitoring System. The foundation classes would perhaps include: class STACK, class QUEUE, class TRACE and class PORT. These foundation classes are services that are application content independent. They are the most general kind of class.

Foundation classes include windows and other GUI classes, data management classes, and dates and times. Foundation classes are often delivered as part of language-vendor's object class libraries. They are relatively well explored.

Not so with essential application and essential business classes. They don't come in a shrink-wrapped box and they aren't always immediately obvious. It's not surprising, therefore, that software engineers who are having trouble with object class identification are usually having trouble finding essential application and essential business classes. Often, successful identification and understanding of these classes is a significant predictor of a project's success.

The essential business classes for the Patient Monitoring System could include class PATIENT, class EKG TRACE (which needs to have properties of the foundation class TRACE and the business class EKG) and class PATIENT STAFF ASSIGNMENT.

The simplest way to find essential business classes is to observe the environment that the system will run in. It will help you identify the real world objects that will need to be abstracted by the object classes you will be building. If you are identifying object classes by intuition, you will be typically looking for nouns in the environment that are of interest to the system. The doctor realizes that this is a simplification. As far as it goes, however, it is accurate.

If you are comfortable Information

BOOK REVIEW

OBJECT-ORIENTED DESIGN — WITH APPLICATIONS *by Grady Booch*

Reviewed by Bob Howard

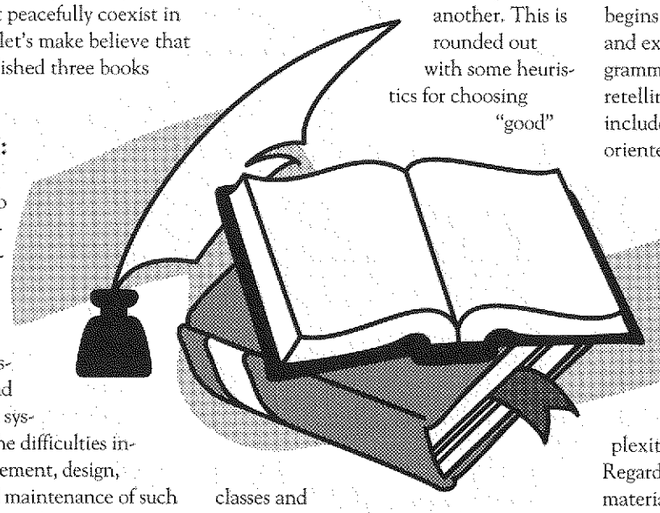
OBJECT ORIENTED DESIGN — WITH APPLICATIONS (Benjamin/Cummings, Redwood City, CA, 1990) is Grady Booch's third book on the subject of object-oriented design. The book is divided into three sections: *Concepts*, *The Method*, and *Applications*. My opinion of these parts varies so widely that my comments on each can't peacefully coexist in a single review. So, let's make believe that Mr. Booch just published three books rather than one.

THE FIRST BOOK: CONCEPTS

When it comes to introducing the concepts of object orientation, it seems that Mr. Booch grows with each retelling. First, a general discussion of the nature and structure of complex systems is presented. The difficulties inherent in the management, design, implementation and maintenance of such systems is reviewed in a clear and concise fashion. These are the problems that led to the creation of structured programming, structured analysis and design, and, eventually, the object model.

Next, Mr. Booch presents one of the best introductions to object orientation I've ever encountered. He not only succeeds in conveying the basic concepts of encapsulation, increased abstraction, inheritance, and polymorphism, but extends this discussion to some of the latest issues such as concurrency and persistence. All of this is presented, to some extent, in the context of programming paradigms in general.

These high-level abstractions are made concrete when the book converts them to their derivative language features of classes, objects, messages, and their relationships to one another. This is rounded out with some heuristics for choosing "good" classes and messaging schemes and an overview of several object-oriented analysis techniques. Since this is a book review, not a personal diatribe, I'll spare you my opinions on the usefulness of current analysis techniques.



In general, if you are new to object orientation, or you'd like to gain more insight into its historical roots and core concepts, this is an excellent overview.

THE SECOND BOOK: THE METHOD

The section on object-oriented design begins with a thorough introduction to, and exposition of, Mr. Booch's object diagramming techniques. With each retelling, he has expanded the notation to include more of the details of an object-oriented design and the ability to capture more of the idiosyncrasies of various object-oriented languages. The level of complexity of the resulting diagrams may lead some readers to wonder if they defeat one of the main purposes of diagramming techniques, the reduction of complexity to easily percieved abstraction. Regardless of your personal opinion, the material is presented in a clear and complete fashion. If you or your organization wish to adopt "Booch-grams", this is the only book for you.

The next sections, *The Process*, pre-

Chapter	Pages	% Of Book
Identifying Classes & Objects	1	18
Identifying the Semantics	1	18
Identifying the Relationships	1.5	27
Implementing Classes & Objects	.75	13

been accounted for in the active and passive components developed in the behavior model. Behavior allocation consists of assigning each of the behavior model objects, or components, to an architectural component.

The nature of the architectural objects depends on their layer. In distributed systems, the top-layer objects are implemented as computers connected with networks. Objects in the next layer might be implemented as processes communicating via pipes. The next layer might be implemented as instances of C++ classes.

A key aspect of this process is that each layer is carried from external model through architecture model before decomposing to the next layer. Effectively, each architectural component becomes its own problem space, to which the design process can be applied.

It should be noted that any design process, by itself, does not confer any certainty or correctness to products. The process can be used equally well for a feasibility study to uncover areas of risk or for final specification of a mission-critical system. Only the experience of the individuals and their organizations can make a product "right."

IMPLICATIONS FOR CHANGE

Two aspects of the process described here have significant implication for changes in the way software is developed. I will describe those changes, and, to see if they are reasonable, compare them with product development practices in other engineering disciplines.

The emphasis on multiple layers of objects, or components, implies that at some point the layers will be developed by assembling components from the next lower layer rather than by writing lines of code in a programming language.

The layers themselves offer a natural way to partition the development process into different roles. That is, different people could work on each layer. People at each layer would be responsible for a narrow band of many applications rather than for almost all of fewer applications. Such a multi-application view is an important prerequisite for developing components that are actually reusable across many applications.

The implication is that one layer of a product could be made by a person writing C++ code while another layer is built by another person writing linker scripts. Is this a radical change?

If we look at software development practices today, two characteristics stand out. One is that only one or two training disciplines (EE and computer science) serve the entire industry. The other is that individuals or a small group can be responsible for everything from design through coding. Compared to these practices, layered design and implementation is a radical change.

Compared to development practices in every other engineering discipline, however, layered development is very much the norm. Electronic hardware is an obvious example of development partitioned by implementation layers (chips, boards, "boxes"). Henry Ford and the Wright brothers designed, built, and operated the first examples of their products. All of those products have changed rapidly and have grown enormously complex. Each industry has handled growing complexity by adding more layers and roles, i.e., different people, to the development process.

Because of the examples of the other engineering disciplines, I feel confident that the layering proposed here will prove to be the path software development will actually take.

SUMMARY

Software product development has some problems with unmanageable complexity, low reliability, and difficulty of maintaining the products. Factors contributing to these problems include lack of modularity and explicit layering in the software structure and lack of a notation for describing an architecture as a set of discrete, communicating components.

Development practices common today directly supporting those structural problems are use of deep decomposition for both analysis and structuring of the system, and vesting responsibility for the entire process, design through implementation, in the same individual, group, and training discipline.

Object-oriented programming languages offer some important help in supporting encapsulation of code and data in the same module. I believe we must go further to realize the kinds of changes needed to resolve the problems with software products. Those further changes include:

- Dividing the design and implementation processes into explicit layers (each layer will be constructed from a different kind of object);
- Designing the communication protocols between objects with the same care and attention as the objects themselves;
- Adding an explicit stage in the design process to identify architectural components;
- Replacing direct, repeated decomposition in analysis and design with a process of shallow decomposition and development of implementation (architectural) components before decomposing to the next level (i.e., only decomposing implementation units).

The notations, techniques, and technology to support these change are beginning to emerge. The next challenge will be integrating the new processes and techniques into existing organizations. ■

Doug Bennett works at DesignWays, which offers training and consulting services aimed at improving processes and organizations that develop software products. He can be reached at DesignWays, P.O.Box 581, Newtown, CT 06470, (203) 425-9511.

MANAGEMENT

Asset management and object-oriented technology

Trevor Hopkins & Brian Warboys

Our aim in this article is to highlight an important change in management style, which is needed to get the best from object-oriented technology. In essence, it is vital to view reusable code as a company asset, in the same way that computer hardware, buildings, and so on are assets. Consequently, companies need software asset management, which is quite different from product management.

This view is especially important for the management of development projects, but its effects permeate the entire management structure. Awareness of this change is needed throughout the company, including development, finance, sales, and marketing sections.

SOFTWARE AS AN ASSET

In typical object-oriented systems, reusable code is presented as a library of classes. These classes are reused in two ways: by using instances of existing classes to build up more complex objects, or by refining the behavior of classes using inheritance. Reusing software in these ways should reduce the development time and the manpower require, since less new code has to be designed, written, and tested.

As reusing software reduces development costs, reusable software components, such as class libraries, are worth money and are therefore company assets. For example, buying a reusable class library costs money; the software purchased is not immediately resalable as a product, but forms an asset from which new products are created. Several companies are now offering for sale libraries of classes in a variety of languages.

To make this approach work, using existing software assets must cost less than buying-in or a complete new development. This requires a change in attitude in developers and an effort on the part of management to ensure that the assets are appropriate for the desired products.

Furthermore, software assets devalue over time, so maintenance of the class libraries is required. This will include bug fixes, improvements to existing classes, and the development of new classes; it is vital to avoid obsolescence in the class library. This implies the dedication of company resources to the development and maintenance of software assets, without immediate input to new products.

Figure 1: Company Software Asset Model

ASSET DEVELOPMENT MODEL

A company's wealth depends on its assets (including software); one view of a company's activities is to regard software production as a "machine" for developing company assets (see Figure 1). The company (and its assets) moves forward in time, "powered" by money brought in by the sales of software products to the marketplace. To manufacture new products, existing assets are reused and new code is developed; new assets may also be purchased. All this consumes the machine's "fuel" (money).

To increase the wealth, and to ensure the company can survive long-term, our assets must be enhanced by maintenance, and by additions to our assets; obviously, we require our assets in 1995 to be greater than they are now! To maximize our assets, we should aim to reuse as much as possible to reduce buy-ins and new development wherever possible, and to enhance our assets for future reuse.

DEVELOPMENT TEAMS

People have always been important for the development of software. This is equally true for object-oriented software development. Indeed, the object-oriented approach encourages teamwork, as there is a closer integration between the roles of analysts, designers, and implementors. (We can regard trained, effective development teams as company assets as well.)

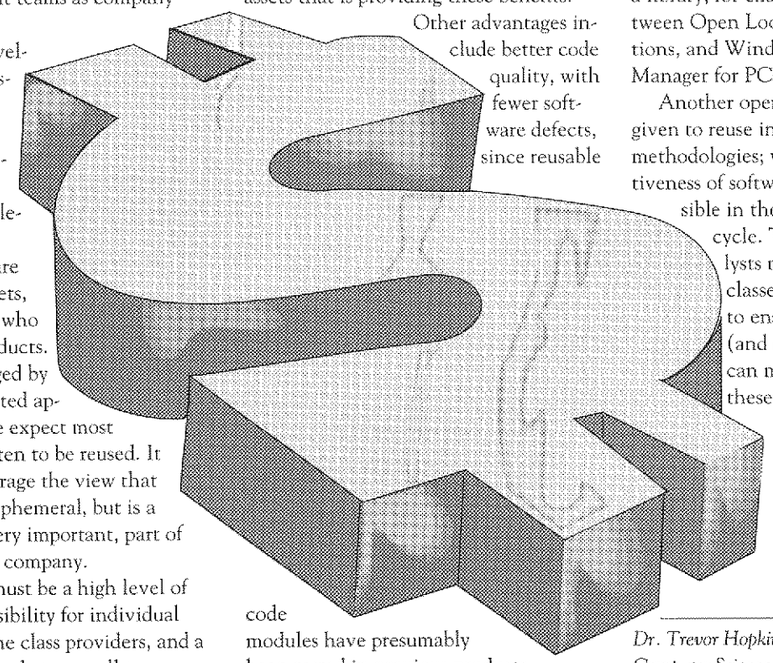
In typical development teams using the object-oriented approach, we observe two kinds of designers/implementors: class providers, who are constructing assets, and integrators, who are building products. This is encouraged by the object-oriented approach, since we expect most of the code written to be reused. It is vital to encourage the view that software is not ephemeral, but is a concrete, and very important, part of the assets of the company.

Also, there must be a high level of personal responsibility for individual classes among the class providers, and a high level of trust between all team members. It is important to foster a culture where reuse of code is expected, because it is easier, faster, and more valuable to the company, rather then writing every part anew.

Since the class libraries are such an important part of the company's assets, it is essential to carefully manage these libraries. A class library "enforcer" is needed: the enforcer's role includes managing the addition of new classes, and new versions of existing classes, ensuring that bugs are reported and fixed, and helping integrators select the most appropriate classes for their application.

CONCLUSIONS

The potential long-term benefits of software reuse utilizing the object-oriented approach are well known. These include a reduction in product cost, thus making the products more competitive, and a reduced time to market for new products, since development times are reduced. In both cases, it is the reuse of assets that is providing these benefits.



Other advantages include better code quality, with fewer software defects, since reusable code modules have presumably been tested in previous products, and vastly improved software maintenance and ease of enhancements.

However, these benefits cannot be realized without a radical change in the way in which software development is viewed within the company, from a management perspective. In particular, management should appreciate the need for short-term investment to build up the software asset base from which new products can be built.

There are a number of technical issues that still need to be addressed before all potential advantages can be realized. For example, the choice of object-oriented programming language(s) is important: different languages have different advantages and disadvantages, and class libraries in one language are rarely useful in another language. The technology to support multilanguage class libraries is still being developed. Also, compatibility with evolving industry standards is important, since these standards will be embodied as classes in a library; for example, the choice between Open Look or Motif for workstations, and Windows 3.0 or Presentation Manager for PCs.

Another open issue is the importance given to reuse in analysis and design methodologies; we must ensure the effectiveness of software reuse as early as possible in the software development cycle. To reuse effectively, analysts need awareness of the classes available in the library to ensure that their analysis (and the subsequent design) can make effective use of these assets. ■

Dr. Trevor Hopkins is a Senior Lecturer in Computer Science at the University of Manchester, U.K. More recently, however, he has moved outside the purely academic sphere, and has been involved in introducing object-oriented techniques into companies such as IBM, International Computers Limited (ICL), the Trustee Savings Bank, Shell Oil, and British Petroleum.

Professor Brian Warboys is an ICL Research Fellow, and (part time) Professor of Software Engineering at the University of Manchester. He has been involved with the management of several large software developments, both within ICL and academia, and has a long-held interest in the technology necessary for the development of large software projects.

≡ Your Turn ≡

How about it? How are we doing? This is the place to send us your message. Tell us what you like and what you don't, what you would like but don't see — and whatever other feedback you'd like to send. Send comments to Marie A. Lenzi, 225 Rector Place, NY, NY 10280; fax to THE HOTLINE, (212)274-0646; or, if you're ordering, send along with your order.

Please check whichever box applies:

	Read entirely	Scanned	Found helpful	Will refer to	Didn't read
Feature	❑	❑	❑	❑	❑
Databases	❑	❑	❑	❑	❑
Objectively Thinking	❑	❑	❑	❑	❑
Environments	❑	❑	❑	❑	❑
On Managing	❑	❑	❑	❑	❑
Dear Doctor	❑	❑	❑	❑	❑
FYI	❑	❑	❑	❑	❑
Product News	❑	❑	❑	❑	❑
Calendar	❑	❑	❑	❑	❑

Please rate the following **1** (dislike or strongly disagree) to **10** (like or strongly agree):

Presentation of material __ Comments: _____

Accuracy of material __ Comments: _____

Overall helpfulness in your job __ Comments: _____

What else would you like to see in THE HOTLINE? _____

Any other comments? _____

_____

_____

Name, title & phone (optional): _____

hotline on OBJECT-ORIENTED *technology*

❑ **Yes,** plug me into the latest thinking and developments in object-oriented technology. Enter me as a subscriber at the term marked below and rush me the current issue. This is a risk-free offer — I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues) ❑ $249

2 years (24 issues) ❑ $478 *(save $20)*

Back issues @ $25 each *($27.50 foreign) — see Page 18:* Vol.1, Nos. _____

(outside US add $30 per year for air service)

❑ **Phone order**
Call Subscriber Service
(212)274-0640

❑ **Fax order**
Fill out term and name/address information, then fax to:
(212)274-0646

❑ **Send me a copy of**
THE INTERNATIONAL OOP DIRECTORY
@ $62 ($69 foreign)
(all directory orders must be prepaid; fax credit card information or mail to SIGS, 588 Broadway, #604, NYC 10012 make check payable to OOP Directory. Foreign orders must be in US dollars drawn on US bank.)

D0KD

❑ **Bill me**
❑ **Check enclosed**
❑ **Credit card orders**
❑ MasterCard
❑ Visa

Mail (and make out check, if applicable) to:
THE HOTLINE
Subscriber Service
P.O. Box 3000, Dept. HOT
Denville, NJ 07834
(foreign orders must be prepaid in US dollars drawn on a US bank)

Card# Expiration Date

Signature

Name

Company

Street/Building#

City/Province

ST/Zip/Country

Telephone

D0KD

Send me a complimentary copy of your related publication(s):

❑ THE JOURNAL OF OBJECT-ORIENTED PROGRAMMING

❑ THE C++ REPORT

Call Subscriber Service
(212)274-0640
or send to
SIGS, 588 Broadway #604
New York, NY 10012

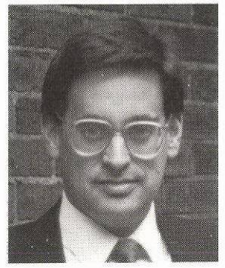
D0KD

hotline on OBJECT-ORIENTED *technology*

VOLUME 1, NUMBER 12 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES OCTOBER 1990

Crown jewels: the value of object-oriented technology in MIS departments

The "crown jewels" of a traditional MIS department are the large libraries of code that perform functions critical to the operation of their business.



Jeffrey Bonar

These libraries typically are written in COBOL, involve hundreds of thousands of lines of code, and rely heavily on mainframe-based databases. They are ideal candidates for creating object-oriented class libraries. In this article, I discuss why this should be done and how to approach such a project.

In a corporate MIS department, the key interest in object-oriented technology is the potential of significant reuse in software development efforts. Corporate software development groups are an ideal place for widespread use of reusable class libraries. Such development groups meet three important criteria for taking the lead in reusable classes: (1) they develop many different applications based on the same back-end capabilities, (2) they have the strongest interest in reusing the software that drives their business, and (3) they have the crown jewel code as a starting point. I believe existing crown jewel code, suitably encapsulated, will be the basis of these libraries.

CORPORATE MIS CROWN JEWELS

Why do I say the "crown jewels"? Why are these code libraries so important? They are a codification of everything the MIS department has learned about the corporation's business. The value of this codification cannot be overestimated. An MIS department's focus is improving the productivity of the core business. The hardest part of this is not, for example, storing invoices in a database, but storing those invoices in exactly the way that optimizes customer payment of those invoices. Typically, the MIS department has taken years to refine their informational capture of the core business — years representing not just design, coding, and maintenance, but also interaction with line workers, middle managers, and top management. A dominant feature of current MIS code libraries is the extent to which they have been honed to support corporate needs.

These code libraries are not without flaws. Mostly, they are based on old technology *continued on page 3 ...*

IN THIS ISSUE

FEATURE — Crown jewels: the value of object-oriented technology in MIS departments *Jeffrey Bonar*	1	17	DEAR DOCTOR — Questions & answers *Steven Weiss*
DATABASES — Why is an object database faster than a relational system? *Mary E.S. Loomis*	6	19	FYI — Industry publication excerpts
ENVIRONMENTS — Visual programming object language environments: a revolutionary approach *Sanford B. Klausner*	9	22	PRODUCT ANNOUNCEMENTS
OBJECTIVELY THINKING — Hype: necessarily (or a necessary) evil? *Jim Adams*	12	23	CALENDAR
ON MANAGING — Déjà Vu *Marie A. Lenzi*	14	24	YOUR TURN — Reader feedback

FROM THE EDITOR

Welcome to the new, and hopefully improved, HOTLINE ON OBJECT-ORIENTED TECHNOLOGY. You may have noticed changes in THE HOTLINE over the last several issues. Those changes were in preparation for the revamp that you see before you now. **Susan Culligan**, **Elisa Varian**, and I have spent a good deal of time reworking both the form and content of the publication; we hope it serves you well. There's a likes/dislikes questionnaire on the back cover — please use it! The only way we know if we're providing you with the information you need is if you tell us.

And again, I encourage letters to the editor. I don't agree with all the articles I print — I expect some of you take exception to certain of them as well. Please communicate this to me. I will print your letters.

The October feature article is **Jeff Bonar**'s "Object-oriented crown jewels." Jeff addresses the huge investment in code libraries and relational databases maintained by traditional MIS shops. He discusses why these libraries are ideal candidates for the creation of object-oriented class libraries, why they should be converted, and how to convert them.

I would like to welcome **Mary Loomis** of Versant as the new database representative on THE HOTLINE's advisory board. Mary is a columnist for our sibling publication, THE JOURNAL OF OBJECT-ORIENTED PROGRAMMING, and this month she talks about why object-oriented databases are faster than relational systems.

Jim Adams of Borland has tackled the "hype" issue surrounding the marketing of object orientation. I agree with his attitude: view a product on its value alone; however, I don't know if I agree with his approach. Too many people believe too much of what they read and hear. I feel that the media has a bit more responsibility than they are willing to accept. I would like to know how you feel about the hype surrounding object-oriented technology. Write to me in care of SIGS Publications.

Sandy Klausner of V4th discusses concepts and approaches to visual programming. Many of the manufacturers of object-oriented technology are doing serious research in visual programming. This just might be the next logical progression beyond objects.

And finally, I have taken the opportunity to share some of my experience with an attempt to implement a major new technology on the premise that those who don't learn from history are doomed to repeat it.

Advisory Board

Jim Anderson, *Digitalk, Inc.*
Grady Booch, *Rational*
Larry Constantine, *Consultant*
Adele Goldberg, *ParcPlace Systems*
Mary E.S. Loomis, *Versant Object Technology*
Bertrand Meyer, *Interactive Software Engineering*
Reed Phillips, *Knowledge Systems Corp.*
Trygve Reenskaug, *Taskon A/S*
Bjarne Stroustrup, *AT&T Bell Labs*
Dave Thomas, *Object Technology Int'l*
Zack Urlocker, *Borland International*
Steven Weiss, *Wayland Systems*

Editor
Marie A. Lenzi

SIGS Publications, Inc.
Richard P. Friedman, Group Publisher
Margherita Monck, General Manager
Elisa Varian, Production Manager
Susan Culligan, Art Director
Melanie Younossi, Marketing Manager
Gregory Tsichlis, Circulation Manager
Kathleen Canning, Customer Service

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012 (212)274-0640.

© Copyright 1990 SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher.
Mailed First Class. Subscription rate—1 year (12 issues) $249. Foreign and Canada $279. Single copy $25.

POSTMASTER: Send address changes & subscription orders to THE HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville NJ 07834.

Submit editorial correspondence to Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280. (212)945-9447.

CAD Information Systems (Cadis) has released **Cadis-Fasteners,** a directory that sits on top of an object-oriented database and lists government and industry standards specifications for commonly used mechanical fasteners. Because the Cadis-Fastener directory is built on an object-oriented database, any fasteners file in the directory can be incorporated in a CAD/CAM file directly via industry export formats. Cadis-Fasteners is available now and is priced at $8,175 per CPU subscription.

Cadis-Fasteners
CAD Information Systems
6551 S. Revere Pkwy.
Suite 220
Englewood, CO 80111
301 799-1311

The V4th project is organized to implement a visual programming object language environment that will surpass conventional software tools. V4th uses a visual declarative programming style that provides a framework for a user to convert cognitive thoughts into computational implementations. The scope of V4th is as general as C++ and is applicable to a wide range of applications including data processing, office automation, hypermedia and embedded software.

V4th is being developed initially for UNIX workstations. The language environment is being integrated with three base server technologies including an imaging model, object database, and hypertext database. A Query-By-Example (QBE) SQL relational API provides a standard interface to relational databases. An information network API provides a data highway for a future electronic large-scale reusable software market system. V4th applications are portable to other target CPU environments through C-compatible kernel consisting of a "RISC" set of procedures.

The implementation plan includes a business model for optimal technology proliferation consisting of strategic partnerships with leading workstation vendors. Each vendor contributes financial and other resources to support the project's implementation stage. In exchange for this advance support, each vendor receives a license to integrate the V4th technology into their workstation hardware. This model distributes the implementation risk among the licensees while lowering the market acceptance risk of new paradigm through broad industry support.

Industry members interested in participating in this project may contact Sandy Klausner at V4th. An animation prototype will be demonstrated at OOPSLA at the Les Suites hotel by appointment only.

V4th Company
V4th Company
2290 Valerie Court
Campbell, CA 95008
408 374-1409

Send Product Announcements to Marie A. Lenzi, 225 Rector Place, No. 12N, New York, NY 10280. Include company name, address, and phone number.

Calendar

THE HOTLINE CALENDAR presents conferences and meetings that focus exclusively on object-oriented technology. To have a meeting or conference listed, please send the dates, conference name and location, sponsor(s) and contact name and telephone number to the Editor: Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280.

October 3–5	October 18–19	October 21–25	November 6–9	November 8
OOP vs. AI	**World of O-O Applications**	**OOPSLA '90**	**Software Development – Fall**	**O-O Structural Design**
San Francisco, CA Sponsor: International Association of Knowledge Engineers. Contact Joan Scaffidi, (301)231-7826	Amsterdam, Neth. Sponsors: Object Management Groups & Patricia Seybold's Office Computing Group. Contact PSOCG (617)742-5200	Ottowa, Can. Cosponsors: ACM & ECOOP. Contact ACM (212)869-7440	Boston, MA Sponsor: Miller & Freeman Conferences Contact Angela Hoyt (415)267-7649	Washington, DC Sponsor: Digital Consulting, Inc. Contact Sue Howard (508)470-3870

Product Announcements

Product Announcements is a service to the readers of THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY; it is neither a recommendation nor an endorsement of any product discussed.

Objectworks\ Smalltalk, Release 4

ParcPlace Systems
1550 Plymouth Street
Mountain View, CA 94043
415 691-6700

ParcPlace Systems announced **Release 4** of **Objectworks\Smalltalk**. The new release is designed to meet the demands of corporate software developers for creating customized, graphic applications for heterogeneous computing environments. Significant new features include integration with standard windowing systems, true color, the Smalltalk Portable Imaging Model (SPIM), incremental storage reclamation, and support for international applications.

Release 4 extends the capabilities of version 2.5 introduced in June 1989. Objectworks\ Smalltalk is the only programming system in the market with Portable Objects, allowing the creation of graphical application that are binary-portable across multiple industry standard computer, operating systems, and windowing systems.

Application can run unchanged on IBM PS/2 and compatibles, Apple Macintosh, Sun SPARCstations, DECstations, IBM RISC System\6000, HP 9000 Series 300 and Apollo workstations without recompiling any code, using native operating and windowing systems.

Other features include 24 bit-per-pixel capabilities for developing true color applications and, addressing the increasingly global business environment, depiction of character sets as 16 bits each, which allows mixing of text in any national language without the need to modify underlying code, provided that the printer, keyboard and display support the national-language characters.

Pricing and upgrade policy will be announced at a later date. It will be available during the fourth quarter of 1990.

Spinnaker Plus

Spinnaker Software
201 Broadway
Cambridge, MA 02139
800 826-0706

Egghead Inc.'s Project Embryo product program has picked up Spinnaker Software Corp.'s object-oriented hypermedia programing software, **Plus**, as its first corporate application. Spinnaker Plus is an object-oriented hypermedia programming environment designed for corporate developers that facilitates the development of customized software applications. The Macintosh edition of the software currently lists for $199.95. Windows and PM versions (to be released later this summer) will list for $495, according to Spinnaker.

Tool Book

Asymetrix, Corp.
P.O. Box 40419
Bellevue, WA 98004
206 455-3071

Tool Book is the first product from Asymetrix (Microsoft co-founder Paul Allen's company). Tool Book is an object-oriented development system providing programming tools to novice programmers, taking advantage of the graphical user interface provided by Windows 3.0.

Santa Cruz buys HCR

UNIX WORLD, July 1990, reports that the Santa Cruz Operation (SCO) announced plans to buy HCR Corp., a leading Canadian UNIX software company. Once the proposal receives final legal clearance and wins shareholder approval, HCR will become an independent SCO subsidiary and will be known as SCO Canada Inc. Founded in 1976, HCR sells a variety of UNIX programming tools, including HCR C++ ... and dbxtra, a debugging and programming development tool. The company will take on much of the responsibility for SCO's development of programming tools. HCR also provides custom UNIX software development services to major clients. HCR's president, Michael Tilson, will continue to run operations and becomes SCO vice president.

continued from page 1

— including obsolete database technologies, obsolete coding practices, and obsolete computing paradigms. They often are batch oriented and include large proprietary components that made sense when they were implemented in the early 1980s but could now be replaced with commercial equivalents at a great cost saving. The biggest weakness of these systems is that they include many years of patches and upgrades, making them difficult to maintain and improve.

Object-oriented technology can enhance the strengths and resolve the weaknesses of the crown jewel code. First, object-oriented technology can effectively provide new code to facilitate interfaces to the crown jewel code. The success of a project to marry object-oriented technology and the crown jewel code rests on the ability to build effective encapsulations of that code. I am using the term "encapsulation" in the traditional object-oriented sense of building classes that represent and provide access to critical functionality. What is being encapsulated, in this case, is the crown jewel code. The core of the crown jewel code can be combined with modern and flexible object-oriented programmatic interfaces. The business knowledge is captured in the crown jewels; the ties to modern software development are provided by the object-oriented technology.

The critical issue is how to facilitate the reuse of the crown jewel code. Reuse in its current form requires mainframe wizards, intimately familiar with the underlying database (IMS, CODASYL, DB2, etc.) and communication technologies (LU 2, LU 6.2 APPC, CICS, etc.) explicitly woven into current implementations. With the judicious use of object-oriented technology, understanding of the database and communication technologies can be hidden in class implementations, leaving relatively simple class protocols for use by day-to-day programmers.

These new crown jewel classes are easily used by the MIS organization. They provide a company standard interface to the crown jewel functionality, allowing the MIS organization to divide up effectively. The back-end wizards maintain and enhance the class implementations, focusing on performance and flexibility. They do not worry about how those classes are being used in day-to-day operations. On the other hand, application developers can focus on meeting the needs of end users, without worrying about maintenance and performance of the back-end. Furthermore, as long as the class protocol is left unchanged, the implementors of the classes are free to upgrade and enhance the implementation. For example, the class implementors can move from an old database technology to a relational system or distributed servers without disturbing the class users. Similarly, the class users are free to improve end-user productivity with graphic user interfaces, without needing approval and understanding from the class implementors. Effectively, the class protocol becomes a contract that frees both groups to focus on what they really care about.

DESIGNING O-O ENCAPSULATIONS OF EXISTING CODE

There is an enormous potential in object-oriented encapsulations of the crown jewel code. How should such encapsulations be designed? Start with the actual business practices of the organization. What are the units of work and performance recognized in day-to-day organizational practice? An insurance business, for example, might be organized around filling out complex, multipart forms. An investment business, on the other hand, is based on many small transactions involving purchase and sale of financial instruments while crediting and debiting the proper accounts. A factory, as a third example, is organized around the resource requirements, production capacity, and output of factory floor equipment.

> ***"The critical issue is how to facilitate reuse of the crown jewel code. Reuse in its current form requires mainframe wizards, intimately familiar with the underlying technologies."***

Design of the crown jewel classes should begin with this kind of broad understanding and appreciation of a business. The classes implemented in the encapsulation should be understandable, at least in a general way, to anyone who understands the core business. Consider some examples from our customers:

- Form, Policy, and Customer classes for an insurance company;
- Transaction, Instrument, and Account classes for an investment firm; and
- Machine, RawMaterial, BillOfGoods classes for a factory.

A set of Collection classes — aggregate data structures like linked lists, sets, arrays, etc. all sharing a common messaging protocol – is often used as a model of reusable class libraries. This is unfortunate; the concerns in the design of reusable Collection classes are too closely tied to the detailed efficiency concerns of a working programmer. Encapsulation of crown jewel

classes should be like understanding your business, not like understanding data structure performance.

Just as crown jewel class design should be divorced from the traditional concerns of programmers, it should also be divorced from the current implementation of the crown jewel code. Classes that represent Customers and Invoices make sense; classes that represent CICS conversations or SQL queries make less sense. Such classes might be useful to the implementors of the crown jewel classes, but they are at too low a level for people developing actual applications.

In summary, the organization of the crown jewel classes and protocols should be based on how the people in a business think about the business. The class organization should not be based on the implementation details of current object-oriented programming products, nor should it be based on the current implementation of crown jewel code.

> ***"There's a certain snobbish disdain for these languages in the object-oriented community. We need to stamp out this disdain ... if object-oriented technology is to be more than an oddity in corporate applications development."***

IMPLEMENTING O-O ENCAPSULATIONS OF CROWN JEWEL CODE

How should an object-oriented encapsulation of crown jewel code be implemented? There are a host of object-oriented technologies available — how should they be judged for undertaking such a project? An object-oriented language or environment must support development teams and provide connectivity with standard programming languages if it is to successfully encapsulate crown jewel code. I explain each requirement in detail.

Requirement 1: support for development teams

The first requirement is that classes produced must be easily shared and managed, in and across development teams. There must be ways to share class definitions on a widely accessible server. It must be possible to have different members of the team responsible for different classes. It is important that these class definitions be manageable with standard source code control systems or their equivalent. The large monolithic environments and multimegabyte image files typical of object-oriented programming in a more formative stage are only acceptable for prototypes and training, not for deploying cross-enterprise systems.

Requirement 2: calling to/from standard languages

The second requirement is the ability to easily link with C, COBOL, FORTRAN, BASIC, and all the other languages where large investments have been made in source code. The crown jewels are implemented in these languages and it is important that an object-oriented programming system be able to easily and flexibly link to this code. We in the object-oriented community need to tie tools to the hundreds and thousands of person-years of development effort. In practice, this means powerful interconnection between traditional languages and object-oriented programming systems.

There are three important aspects to this connectivity:

1. It must be possible to call routines written in traditional languages from the object-oriented programming system. Such an ability makes it relatively easy to tie to standard subroutine libraries and application programmer interfaces (APIs). Without this ability, it is next to impossible to tie to code that is not so cleanly organized.

2. There must be APIs for the standard languages that allow calling the object-oriented programming system. It should be possible to, for example, send a message to an object from a line of COBOL code. It is acceptable for this to be clumsy, but it must be possible. The critical issue here is that the object-oriented programming code cannot always be in control nor can it be waiting for the standard code to return from a subroutine call. For example, much of crown jewel code is time critical and must be written in a low-level language like C. In other cases, there may be minutes between the initiation of a crown jewel activity and response from this activity. In either case, it is important that the crown jewel code can notify the object-oriented programming code independent of activities of that object-oriented programming code.

3. There must be a way to share data structures between object-oriented programming code and standard code. If the calling interfaces discussed in items 1 and 2 can only pass primitive data types (integer, real, character), then passing data becomes a critical bottleneck. There must be a way to share standard aggregate data structures (arrays, C structures, COBOL records, etc.) with the object-oriented programming system. In particular, these data types should be passable by reference (by address) so they do not need to be packed and unpacked at each end.

Large Companies Supporting OOT

Even though most end-users do not realize it, many software vendors are already embracing OOP languages and database structures. However, most users, one step removed from the underlying code, still feel that OOP is in its infancy, and do not plan on buying actual OOP packages until the mid- 1990s. ... Among vendors, the early adopters of OOP are using it to build icon-oriented graphical user interfaces or "hot-linked" applications that pass data back and forth. They include Hewlett-Packard Co., Apple Computer Inc., and Microsoft Corp.'s EDS division. ... However, the steep learning curve still puts off many potential users. "One of the real problems is that the right language for the users doesn't really exist," Hellenack [Les, director of new software technologies at International Data Corp.] explained. "What you're seeing is a staging of OOP technology. There are object-oriented programming languages, and object-oriented databases, but the analysis and design tools aren't there yet." ... However, Hellenack said he believes that the OOP revolution is not far off. Driven by a need to reduce development costs — and allow users to write their own applications — IS departments should be shopping for OOP products by the time 1992 rolls around, he said.

Source: "Firms Move into the OOP Lane,"
***COMPUTERWORLD**, Jean S. Bozman, July 23, 1990.*

New Wave [from Hewlett-Packard] has features that Windows does not, including automated "agents" that can be created to perform office tasks, extensive help features, and computer-based training lessons, and a remarkable facility for getting many different software applications to work together as if they had been built that way. ... New Wave, which will cost $195 when it becomes available in August, provides a somewhat better graphical interface than Windows 3.0. With Windows, the user still has to have some grasp of the way DOS handles computer files; with New Wave, DOS seems to disappear entirely. ...On a deeper level, New Wave is significant because it uses a programming technique called object-orientation.

Source: "A Program that Harnesses Other Software's Strengths,"
***OAKLAND TRIBUNE**, Peter H. Lewis, July 2, 1990.*

Anti-hype

Object-oriented programming holds a great deal of promise, especially in development situations where each module is created and compiled separately. Although some may welcome its intuitive approach to programming, OOP is not necessarily for everyone. Even many experienced programmers find it too rigid and complex for their own use. The fact remains that for many people, the logic and structure of most programming languages continue to be too complex and uncomfortable. For the average computer user, even these relatively newer approaches seem just as difficult to understand as the techniques used by the mainframe programmers and technicians of a generation or two ago.

Source: "Democratic: Amiga in the New Age,"
***AMIGAWORLD**, Louis R. Wallace, August 1990.*

As the head of Cleveland-based Morgenthaler Ventures' outpost in San Francisco, [Gary] Morgenthaler is looking for new territory to conquer. "I don't see that many fundamentally new technologies," said the man of a thousand business plans. Many of the buzzwords that have captivated the software world, leave Morgenthaler cold. Object-oriented technology [is one of those]. Despite its obvious technical appeal, Morgenthaler sees a market emerging more along the lines of artificial intelligence than relational database. "I see a niche market for those guys."

Source: "Soft Talk: Hot Lunch," ***COMPUTER SYSTEMS NEWS**,*
Joshua Greenbaum, July 30, 1990.

Newsworthy Revelations

Carleton University has launched a joint Ph.D. program in computer science with the University of Ottawa. ... According to Pugh [John, director of Carleton's School of Computer Science] Carleton is ideal for Ph.D. candidates interested in object-oriented programming (OOP) research. ... Students pursuing OOP studies have access to a laboratory containing three Sun 3/140 workstations, a 3/150 workstation with 1.18 GB of hard disk space, and five Macintosh II workstations. ... So far, two students have entered the program. Another four will be admitted in the fall, said Pugh. Based on labor market projections, none of them should encounter difficulties in landing academic, public or private sector positions upon graduation, he predicted. "In fact, graduates at any level shouldn't have any problems getting a job."

Source: "Schools Offer Computer Science Ph.D.,"
***COMPUTING CANADA**, John Eberlee, July 19, 1990.*

Languages & Tools

The key issue today is to get object-oriented Cobol out of standards committees and into practice. Object-oriented Cobol is a breakthrough in software engineering, and this will affect the information systems world in much the same way that structured methodologies and relational databases did in the '70s and '80s, respectively. ... Object-oriented Cobol could cut down new program development time by 70% at large Cobol sites. In other words, 70% of all Cobol application code now written could be reusable as objects invoked by new programs. This has enormous implications for applications maintenance.

Source: "Cobol — The Next Stage," COMPUTERWORLD, Jerome Garfunkel, June 23, 1990.

Bellcore has developed what it said is a highly efficient, object-oriented computer language called Laure that uses logical deduction to solve complex programming problems. "Laure programmers need only describe the software writing result they are seeking, and Laure will determine the best way to reach that goal," said Bellcore researcher Diane Hoffoss. ... Laure organizes facts and related instructional data into sets and then defines specific relationships among all the sets in a program.

Source: "A Logical Language," COMPUTERWORLD, July 30, 1990.

Microsoft Corp. opened the doors to its language development labs last week, providing a glimpse at a long term game plan to add improved optimization and object-oriented programming capabilities to its line of tools. ... To endow its languages and operating systems with object-oriented capabilities, Microsoft is embracing a conceptual model known as the Microsoft Common Object Model. This defines a series of object managers that will be built into languages, applications and operating systems, said Martin O'Riordan, project leader for Microsoft C++. ... In the more immediate future, Microsoft will upgrade its C compiler to support 386- and 486-specific development. This will appear in concert with the introduction of Microsoft's 386-based OS/2 version 2.0, which Microsoft plans to deliver by the end of this year.

Source: "Microsoft Hints at Future Language Plans," PC WEEK, Paul M. Sherer, June 25, 1990.

At Software Development '90 last February, I spoke with the president of a leading C++ translator vendor. He informed me that because of increasing insults to his product, firm and person, he would no longer attend object-oriented-specific trade shows such as OOPSLA and SCOOP. The bickering over C++ got just a bit out of hand. ... Many vendors and users of other object-oriented languages view C++ and its rapid rise in market share as a 500-lb. canary charging through the ranks of the object-oriented community. In other words, despite the limited support for its run-time polymorphism and other deficiencies, C++ has too much momentum to be successfully challenged. ... [However] the object-oriented market is exploding for a number of reasons... Sales of other object-oriented languages such as Smalltalk, Actor and C-talk can be expected to rise far above the levels reached in previous years.

Source: "The Rise of C++: For Richer for Poorer?," COMPUTER LANGUAGE, July 1990.

The hackneyed metaphor for creating software — "building" software programs — has boxed in programmers. The new thinking is in "growing" software — something some kinds of object-oriented programming allows. ...Our terminology sounds like we are in the house construction business. We complete specifications, we assemble components; we build scaffolding. De-bugging smacks of calling the exterminator: the house is OK — there are just some bugs running around in the cracks. ... Growing software is a familiar experience to most Smalltalk developers. ... One useful way to think about Smalltalk development is that it involves changing a living, breathing application whose backbone is defined by the class hierarchy. Creating a new class adds a bare bone to the class hierarchy. Creating a new method adds flesh to the class. If necessary the class can be put on a diet to eliminate unnecessary code. We can perform major surgery to cut out unwanted classes. The underlying organ, the application, is always changing, but it's always alive.

Source: "It's a Growth Industry," CALIFORNIA COMPUTER NEWS, Jim Anderson, July 1990.

Standards

The Object Management Group has released the core of its proposed standards manual, and will be seeking comment on it through July 31. The core consists of three elements — a Reference Model, Abstract Object Model, and Technical Objectives, said Chris Stone, president of the standardization, promotional and educational group of 70 vendors and major users. ... The group's mission is to generate reference implementations of object-oriented technology, promote those implementations as standards by recognized standards bodies, evangelize for object-oriented technology, and educate users and vendors about the technology, said Stone. OMG's support of object-oriented technology comes from an angle that might be considered somewhat peculiar, Stone acknowledges. ... "Programming today is an art form," said Stone. "We have to get it from the grips of impressionism and into a science."

Source: "OMG Releases its Standards Manual," UNIX TODAY, Mitch Wagner, June 25, 1990.

For example, consider an application where crown jewel code produces sales information in an array of five hundred to one thousand records, each with 400 bytes of data. An application developer is to use the object-oriented programming system to create an interactive graphic interface that allows middle managers to browse this data. With only primitive data types, moving the data from the crown jewel code to the object-oriented programming system is a very slow process. With shareable data, all that need be passed is the address of the front of the array and the number of records in the array. Ideally, the object-oriented programming system can treat the records in the array as if they are normal object instances with, perhaps, certain restrictions not required of standard objects.

IMPLEMENTATION OF ENCAPSULATIONS

Beyond specific implementation features, how should a project to encapsulate crown jewel code be managed? What problems should be anticipated? It is critical to start with small, easily managed projects. There is no way a deep understanding of object-oriented technology can be acquired from courses and books. It is important to learn by completing real projects connected with real business problems. Only after gaining experience in this way should you begin development of object-oriented encapsulations of crown jewel code.

As part of the planning for crown jewel code encapsulation, we would budget for a three- to six-month pilot project. This a crucial step allows us to test our encapsulation design against real-world needs. We would be willing to throw away the pilot implementation and the plan that led to it if it is not right. The crown jewel encapsulation will be the backbone of the MIS department for the next decade — it is important to take the time to fully exploit object-oriented technology and learn from the first efforts.

As part of the planning process, it is important to identify which crown jewels need an object-oriented encapsulation and which need to be (eventually) replaced. Initially, all that is needed is the specification of class protocols. The first implementation of these protocols can be implemented in a stop-gap fashion, sacrificing back-end performance. For example, many of our customers are updating their mainframe programs by writing object-oriented graphic user interfaces. The classes representing the mainframe computations actually communicate with the mainframe by reading and writing to a terminal emulator (3270 EHLLAPI). No changes are required to the mainframe-based application program; it thinks it's talking to a dumb terminal. In exchange for the back-end performance loss, an enormous boost in application developer productivity is provided by the clean class protocol defining the back-end capabilities. For now, we accept that the methods for the classes talk to the mainframe through a terminal emulator. Over time, the back-end capabilities will be reimplemented in a more effective way with little or no change to the work done by the application developers.

ONE FINAL CAUTION

The transition to an object-oriented framework for crown jewel code requires learning more than just object-oriented technology. Along with the obvious need to learn new design and coding techniques, it is likely that a crown jewel encapsulation project will become the focus for acquiring a range of new technologies. These technologies include interactive development environments, graphic user interfaces, event-driven interfaces, new workstation operating systems, networking, client-server architectures, and distributed processing. Sensible transition to these new technologies must be planned along with the transition to object-oriented techniques.

CONCLUSIONS

Linking object-oriented technology with large code libraries, the crown jewels of corporate MIS departments, is an ideal marriage. The crown jewel code libraries represent a priceless treasury of business theory and practice. Encapsulation in class hierarchies can modernize the crown jewel code at a relatively low cost. At a tiny fraction of the cost of other approaches, object-oriented technology can deliver the power of established applications with an enormous boost in application development productivity. This article asks for an enormous change in how corporations develop application software, but proposes a manageable pathway for making those changes. It is incumbent on object-oriented technology vendors to make this pathway easier and easier to tread. The corporations that implement such an approach first will define corporate MIS in the 90s. More importantly, they will dominate their industries, truly demonstrating the power of effective information management. ≡≡

Jeffrey Guy Bonar is Chairman and Chief Technical Officer of GUIdance Technologies. Prior to joining GUIdance, Bonar was with the University of Pittsburgh Learning Research and Development Center, where he was a founding member of the Intelligent Systems Studies Program. Bonar can be contacted at GUIdance Technologies, Inc., 800 Vinial Street, Pittsburgh, PA 15212, (412)231-1300 or Fax at (412)231-2076.

DATABASES

Why is an object database faster than a relational system?

Mary E.S. Loomis

Object technology is making big waves in the software world — languages, methodologies, tools, user interfaces, databases, and management techniques are all being affected.

As this is a database column, I'll focus on the data management world, where the ODBMS vendors are establishing their track records.The relational DBMS vendors are taking notice, modifying their products and marketing plans to benefit from the growing interest in objects. Many applications vendors are now evaluating ODBMS, aiming to gain competitive advantage through prompt introduction of object-oriented products. Developers who were unable to use relational systems because of unacceptably slow performance are finding there's a world of difference with ODBMS. Not only is performance significantly better; the interfaces are typically easier to use as well, especially when coupled with an object-oriented programming language.

Mary E.S. Loomis, Ph.D., is Vice President of Versant Object Technology, formerly OBJECT-Sciences Corporation. Loomis was previously R&D Lab Manager in the Data and Language Division of Hewlett-Packard and Director of R&D at GE-Calma. She has written four books on data management, file structures, and data communications and is a regular columnist for THE JOURNAL OF OBJECT-ORIENTED PROGRAMMING. Loomis can be reached at Versant Object Technology, 4700 Bohannon Drive, Menlo Park, CA 95025, or through email at mloomis@osc.com.

A PLEASANT SURPRISE

The excellent performance of ODBMS may be surprising to those who've weathered the introduction of previous generations of data management technology. When relational DBMSs were introduced in the early 1980s, they were practically unusable for production applications because they were so much slower than the installed network and hierarchic DBMSs (IMS, IDMS, Total, etc.). Initially, the relational products were applied in areas where performance didn't matter very much (decision support systems, prototype applications), but where their flexibility and relative ease of use did.

Over the years, the performance of the relational DBMS products has improved dramatically. They are now used in a wide variety of commercial applications. Nonetheless, their performance has remained unacceptable for other families of applications, including many applications with complex data relationships and very stringent performance requirements. Even though the ODBMS products are relatively young, they are entering the race with excellent performance — good enough for the demanding applications areas of CASE, CAD, GIS, network management, concurrent engineering, and so forth.

I offer an example that helps to show why object database performance can be significantly better than relational database performance. The example will also illustrate that an object database structure (schema) can be more intuitively understandable than a relational database schema for the same application. This more intuitive nature leads to more maintainable software, which has benefits of better quality and easier adaptability to changing business requirements.

The example includes both an object model and a relational model to support a graphics application. Let's first compare the ease of understanding the two models.

THE OBJECT MODEL

Figure 1 illustrates several example PolyLines, commonly used in graphical systems. Each PolyLine is composed of lines connecting points in three-dimensional space. Figure 2 shows a simple object model for a PolyLine. Each PolyLine object is comprised of a list of Points, in the order in which they appear in the line. A method for drawing a PolyLine is part of the class description. Each point lives in three-dimensional space and thus has three coordinates; the PolyLine

FYI

Excerpts from leading industry publications on aspects of object-oriented technology

Applications

By increasing productivity and shortening the length of time needed to develop applications, object-oriented database management systems (OODBMSs) are ready to fill the database void for CAD users. Once new applications are developed and existing ones are converted, the paybacks from the evolution to OODBMS-based CAD tools will be substantial for both developing and maintaining code. ... At the same time, the object-oriented DBMS must operate at performance levels often two orders of magnitude faster than existing DBMS technology. This tension between the need for a full suite of traditional DBMS capabilities and very high performance makes the question of implementation of architecture a key one for object-oriented DBMS design. The basic question most CAD users will ask is "How can you get it to go that fast?" ... Sharing, distribution and high performance will be the initial reasons CAD users turn to object-oriented DBMSs, but the higher level of abstraction will be the reason object-oriented technologies come to dominate. For designers, object-oriented technology makes it easier to model real world entities.

Source: "Object-Oriented DBMS: Signs of the Future," MICROCAD NEWS, Tom Atwood, July 1990, pp. 22-24.

Seriously, it appears to me that a truly object-oriented database is essential for a powerful hypermedia system. And I'm not talking about object-oriented-like veneers around a relational database, which would be too slow. The very nature of relational databases doesn't seem to map well to the structure of truly object-oriented architectures. Simulating objects in a relational database would entail multi-multi joins or cascading triggers (for referential integrity) or both, bringing a relational database to its knees in a hurry.

Source: THE PROGRAMMER'S JOURNAL, August 1990.

Computer scientists at Sandia National Laboratories have developed an object-oriented programming environment designed to speed the process of developing software for robots and improve programmer productivity, among other benefits. The Robot Independent Programming Environment (RIPE) allows programmers to extend software code developed in one robotic application to other applications, scientists said. RIPE has been used in Sandia's laboratories in a demonstration of robotic cask handling operations for possible use in nuclear waste facilities.

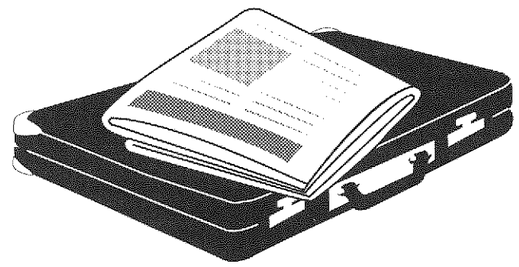
Source: "RIPE for Robots," COMPUTERWORLD, June 25, 1990.

Carousel Mediaworks...[of] Honeoye, NY...develops training programs on personal computers using multimedia authoring software for clients such as Eastman Kodak Co. and Xerox Corp. ... Computer-based training is finally coming of age thanks to low-cost PCs and object-oriented programming (OOP) software that turns course developers into multimedia producers. ... OOP technology-based software authoring tools are a "monumental step forward in generating computer-based training programs," McElroy [Denny, founder and president] said. "Now the subject matter expert has the tools that allow him to sit down and do a presentation just like an author uses a word processing program."

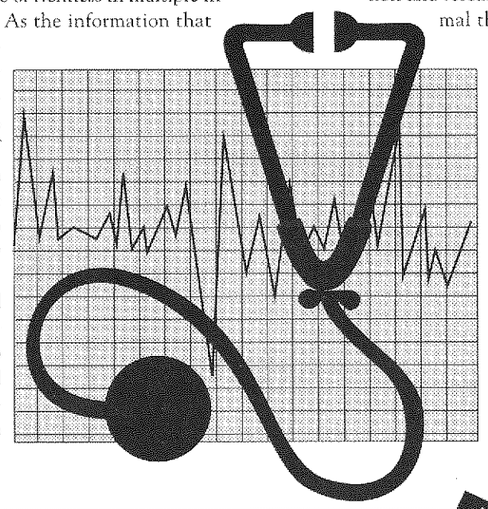
Source: "Cutting Costs with Computer-based Training," COMPUTERWORLD, Michael Alexander, July 30, 1990.

According to Robert Martin, a vice-president at Ontologic, an object-database integrates an object programming language (usually C++) with database functionality, thereby fulfilling the requirements to build high-performance database applications with well-structured, easily maintained applications. Such databases save objects on the disk so that they can be used in future sessions or shared by many users, and they also provide applications programs with an abstract interface to data, enabling programmers to change or reorganize that database without changing any application code. Martin claims that object-oriented databases have one major advantage over relational databases: namely that relational databases aren't flexible as they have to store all data in fixed length records — all applications data must be converted to these records making relational databases slow and prone to error.

Source: "Object Databases are Here: Is This the End for Relational Models?," UNIGRAM X, Katy Ring, June 1990, p. 4



only a part or (disjoint) parts of the tree will be of interest. Add another degree of richness in multiple inheritance environments. As the information that completely defines or describes a class is typically fragmented across the terrain, engineers tends to spend some portion of their engineering time in a role not unlike a private investigator running down the fact that is currently important.



I think technology will provide a healthy component to the solution to this problem. We will need browsers that present the class hierarchy in ways that facilitate knowledge transfer and assimilation and research. Research is more formal than browsing. In a sense, I am proposing a form of CASE for the class hierarchy. They are becoming too complicated to navigate and understand manually.

Work is underway in this area. For example, Bertrand Meyer is working on a flattener, a tool which presents details about an object and its lineage at once. It's a step (maybe a leap) in the right direction.≡ ≡

hotline on OBJECT-ORIENTED technology

Back issues available

Please mark the issues you wish to receive by volume & number on coupon on back page

Reprints available on request — call Kathy McKendrick at Reprint Resources (215)643-9140

Vol.1,No.11/Sep. 1990 ≡ FEATURE: Introducing OOPS through design — it's all in the words•*Seth Copen Goldstein* ≡ DOWN UNDER : The Australian object-oriented scene • *Dr. B. Henderson-Sellers* ≡ Objects on tour (Review of Borland's OOP World Tour)•*Marie A. Lenzi* ≡ ANALYSIS & DESIGN: Design principles for modular product development•*Douglas W. Bennett* ≡ MANAGEMENT: Asset management and object-oriented technology•*Trevor Hopkins & Brian Wyerboys*

Vol.1,No.10/Aug. 1990 ≡ FEATURE: WyCASH+: An application built within an OOP environment•*Paul McCullough & Nancy Deshler* ≡ DATABASES: Mathematics and methodology•*Robert R. Martin* ≡ PROJECT MANAGEMENT: Talking to project managers: Organizing for reuse•*Adele Goldberg & Kenny Rubin* ≡ STANDARDS: CALS Expo–89 Conference•*Alan Purdy*

Vol.1,No.9/July 1990 ≡ FEATURE: Cognos and Eiffel: a cautionary tale•*Burton Leathers* ≡ What's in an object anyway?•*Andrew J. Novobilski* ≡ LANGUAGES & ENVIRONMENTS: The impact of Windows 3.0 on the OOP market•*Zack Urlocker* ≡ PROJECT MANAGEMENT: Talking to project managers: Introducing reuse into the development process•*Adele Goldberg & Kenny Rubin*

Vol.1,No.8/June 1990 ≡ FEATURE: A taste of the USENIX C++ Conference•*Mary H. Wells* ≡ DATABASES: Comparing relational and object database management systems part 2 — Composites, flexibility, and extensibility•*Robert R. Martin* ≡ PROJECT MANAGEMENT: Talking to project managers: Case studies in prototyping, part 3•*Adele Goldberg & Kenny Rubin* ≡ ANALYSIS & DESIGN: Object-oriented requirements analysis•*Edward V. Berard*

Vol.1,No.7/May 1990 ≡ FEATURE: Object-oriented development: not business as usual•*Mark Lorenz* ≡ DATABASES: Comparing relational and object database management systems part 1 — Single paradigm programming is the key difference•*Robert R. Martin* ≡ PROJECT MANAGEMENT: Talking to project managers: Case studies in prototyping, part 2•*Adele Goldberg & Kenny Rubin* ≡ ANALYSIS & DESIGN: Understanding the recursive/parallel lifecycle•*Edward V. Berard*

Vol.1,No.6/April 1990 ≡ FEATURE: Lifecycle approaches•*Edward V. Berard* ≡ DATABASES: A data manipulation language for C++ developers — a proposal•*Thomas Atwood* ≡ PROJECT MANAGEMENT: Case studies in prototyping, part 1•*Adele Goldberg & Kenny Rubin* ≡ ANALYSIS & DESIGN: I already learned in kindergarten all I needed to know about object-oriented design•*Niranjan Ramakrishnan*

Vol.1,No.5/March 1990 ≡ FEATURE: Objects of desire•*Christopher M. Stone* ≡ DATABASES: OMG: The state of the object-oriented DBMS market•*Tom Atwood* ≡ PROJECT MANAGEMENT: Talking to project managers…•*Adele Goldberg & Kenny Rubin* ≡ INTERNATIONAL BEAT: The Australian object-oriented scene • *Dr. B. Henderson-Sellers*

Vol.1,No.4/February 1990 ≡ FEATURE: Greed: a complete example•*Tom Love* ≡ DATABASES: OMG: an industry push for standards•*Tom Atwood* ≡ PROJECT MANAGEMENT: Talking to project managers…•*Adele Goldberg & Kenny Rubin* ≡ LANGUAGES & ENVIRONMENTS: OOP in the 1990s•Zack Urlocker

Vol.1,No.3/January 1990 ≡ FEATURE: Adopting object-oriented techniques: management and organizational impacts•*Marie A. Lenzi* ≡ DATABASES: The object-oriented database system manifesto: a consensus from academia•*Tom Atwood* ≡ LANGUAGES & ENVIRONMENTS: Which language to choose, revisited•*Jeff McKenna*; On language wars•*Bjarne Stroustrup*

Vol.1,No.2/December 1989 ≡ FEATURE: OOPSLA '89•*Jeff McKenna* ≡ GUEST ARTICLE: C'ing into the future•*Keith W. Howard* ≡ DATABASES: Standards for OODBMS: a manager's guide•*Tom Atwood* ≡ ANALYSIS & DESIGN: Synthesis: an object-oriented analysis and design method•*Mellir Page-Jones & Steven Weiss* ≡ LANGUAGES & ENVIRONMENTS: From languages to environments•*Zack Urlocker*

Vol.1,No.1/November 1989 (Premiere Issue) ≡ FEATURE: Which object-oriented language should we choose?•*Ken Auer* ≡ DATABASES: An introduction to object-oriented database management systems•*Tom Atwood* ≡ ANALYSIS & DESIGN: Understanding object-oriented analysis•*Sally Shlaer & Stephen J. Mellor* ≡ LANGUAGES & ENVIRONMENTS: Object-oriented programming: myths and realities•*Zack Urlocker*

knows how many points it contains. A database description of these objects using approximate C++ syntax is:

```
    class Point : virtual public pObject
{   public:
            double x,y,z;    }
    class PolyLine : virtual public pObject
    {   public:
                int noOfPoints;
            List<Point> points;
            void drawPoly()
            {   for (int i=0;i<noOfPoints;i++)
drawLine (points[i], points[i+1]);
            }
    }
```

THE RELATIONAL MODEL

A corresponding relational model includes three tables: PolyLine, Point, and PointUse, as follows and as shown in Figure 3.

```
PolyLine (polyLineID, noOfPoints)
    Point (pointID, x, y, z)
    PointUse (polyLineID, pointID, sequence)
```

PolyLines and Points have been assigned identifiers polyLineID and pointID, respectively. These are needed to fulfill the relational requirement that each row of a table must have a unique identifier comprised of attribute values. The several tables are needed to fulfill the relational requirement for flat rows (for example, no row can contain a table of values), and to achieve a design that meets the rules of relational normalization. This example does not include the object-oriented notions of inheritance or containers, and already the relational representation looks less intuitive than the object representation.

DRAWING THE LINE

Now let's compare the object database and relational database activities of finding the points of a PolyLine in sequence, say to support drawing the line. In the object database case, the PolyLine object contains a list of its points in sequential order. The application programmer would simply invoke the drawPoly method, which iterates through the points list noOfPoints times, connecting the points with lines. The next point in the list is the next point of the line.

> ***A fundamental characteristic of a relational table is that the ordering of its rows is immaterial and even sometimes unpredictable.***

By contrast, in the relational case there is no guaranteed correspondence between the order of appearance of the points in the PointUse table and their sequence on the desired PolyLine. A fundamental characteristic of a relational table is that the ordering of its rows is immaterial and even sometimes unpredictable. A temporary structure is needed to store the points of the line in sequence for the drawing process. An appropriate structure is a points array defined as follows.

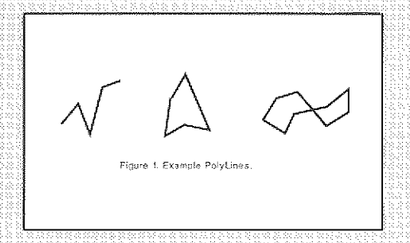


Figure 1. Example PolyLines.

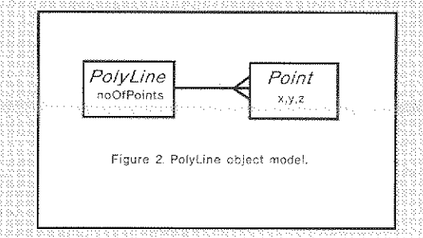


Figure 2. PolyLine object model.

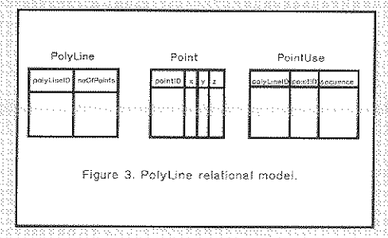


Figure 3. PolyLine relational model.

```
typedef struct (double x,y,z) Point;
    Point points[1000];
```

Obtaining the points in sequence from the relational database can now be specified using SQL and C++ as follows. The desired PolyLine has been assigned a unique identifier in host variable lineNumber. The relational database retrieval and drawing process is:

```
Select PL.noOfPoints
        Into :max
    From PolyLine PL
        Where PL.polyLineID=:lineNumber;
int i=0;
    Select all P.x, P.y, P.z
        Into :points[i++]
        From Point P, PointUse PU
        Where P.pointID=PU.pointID
        and PU.polyLineID=:lineNumber
        Ordered by PU.sequence;
    for (int i=0; i<max; i++) {
        drawLine (points[i], points[i+1]);
    }
```

> ***A relational system's lack of built-in access paths ... is the primary reason that relational database performance can be unacceptable for large classes of applications ...***

It is important to note that two very different kinds of processing are taking place in the relational and object DBMSs, with significant performance implications. The relational case first searches through the table of PolyLines to find the particular one of interest and extracts its noOfPoints into host variable max. Then the PointUse table is joined with the Point table on matching pointID values to select x,y,z values sorted by their sequence in the PolyLine per the pointUse table; the x,y,z values are copied into the temporary points array that was defined above.

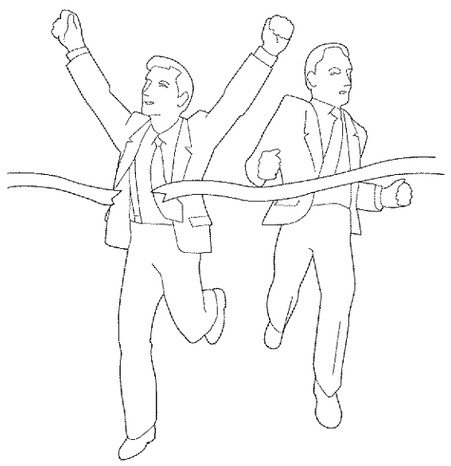
Joining and sorting compete for being the slowest operations possible in relational systems. Both are needed here. Joining two tables requires finding all the pairs of rows from the two relations that have the specified match in attribute values. If one table has Nrows and the other has Mrows, then the join may involve N*M comparisons. The relational systems can cut this number down by using various optimization techniques, including indexes, sorting the rows, and so forth. Still, if the tables are large, performance can be unacceptably slow.

MODEL THE REAL WORLD

Object DBMS performance excellence comes in part from avoiding the concept of joins altogether. The real world's logical structure is modeled directly in the database structure. In this example, the object database stores the list of the points that comprise the PolyLine right in the PolyLine object. Application-defined access paths are built into the object database; they are part of the application's object model. By contrast, a relational system's lack of built-in access paths to match the way that applications want to traverse related data is the primary reason that relational database performance can be unacceptable for large classes of applications, including almost any application that needs to drive high-speed graphics.

Acknowledgment

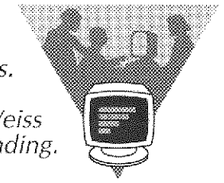
Thanks are extended to Bill Baloglu from Versant's Technical Staff, who contributed many of the ideas in this paper. ≡≡



≡≡DEAR DOCTOR

A forum on object-oriented methods, practices and procedures.

Dr. Steven Weiss attending.



Dear Doctor:

We think that object-oriented methods will help us improve our functional requirements definitions. However, we are unable to move to an object-oriented language. What can we do?

A system that is organized around objects and built with a traditional, non-object-oriented language is object based.

Just as you can write well-structured code in an assembly language (which is inherently unstructured), you can write well-structured object-based systems, with some desirable O-O properties, in a language that has none already built in. Depending on the degree of object-orientedness you strive for, coding may be more labor-intensive, but don't assume that it will be. My own experience has been that an object-based organization for a project may actually be less labor intensive, certainly if testing, debugging and ongoing maintenance are taken into account when you take your labor measurements.

Object-based engineers need to decide how much object orientation they wish to introduce into their inherently non-object-oriented environment. The two most visible characteristics of object orientation are encapsulation and inheritance. An O-O language provides both conveniently. A traditional language provides inheritance with difficulty and encapsulation a little more easily.

In any case, programmers need to understand the object-oriented abstraction clearly. They will be manually crafting constructs that a compiler normally provides.

It's common to see object-based systems that use encapsulation only. The idea has a rich heritage. David Parnas introduced data abstractions and Meilir Page-Jones introduced information clusters over a decade ago. With relatively minor enhancements, Meilir's information clusters can be made to serve as object classes with instances.

Inheritance is a bit more difficult. I don't recommend that you hard code inheritance relationships into any code that you ever contemplate maintaining. The inheritance mechanism must be transparent to the programmer if it is to be an asset. If it isn't it, will become a liability at the worst possible moment — if not during development, undoubtedly during enhancement.

A handful of sites have built generalized message handlers (in the O-O sense). One even did it for COBOL programs. The class hierarchy was defined with tables internal to the handler. It's an approach you might consider if you know that an O-O language is not on your near horizon.

Dear Doctor:

What is the most effective method to educate new company engineers on the objects found in a product? How do you pass the knowledge from senior engineers to new hires?

You have asked a significant question. The problem of transferring knowledge about reusable components has existed since their invention. Objects are among the most sophisticated reusable components we have. They thus are among the most difficult to learn and transfer knowledge about.

I remember my first exposure to the Small-talk class hierarchy. If it is really true, as DeMarco asserts, that the complexity of a document is proportional to the number of fingers needed to read it with, the document that describes the class hierarchy is nearly off the scale. (The browser improved matters, but not as much as I would have liked.)

Why? To understand any arbitrary class in all its nuance, one needs to potentially consider the topic class and all the classes that it inherits from. Occasionally, an engineer will be interested in the whole inheritance tree, sometimes the data component only, sometimes the process component only. At other times,

The Doctor has been a consulting methodologist for over a decade. He invites you to submit your questions on object-oriented technology to

Dear Doctor
1202 108th Ave. East
Puyallup, WA 98372.

It all depends on the environment in which you work and on the people with whom you work, the corporate strategy and objective, whether software is a direct revenue generator or an expense, etc., etc., etc. But, no matter what the problem/solution set is, if the nontechnical issues are ignored, the new technology will not grow.

Managers have operated on the egg side of the "bacon and eggs" mode: the chicken is involved, the pig is committed. The promises will not be realized unless managers take as active a role in the implementation of object-oriented technology as do the developers. All the issues (technical and nontechnical) that were in play with other major new technologies are still valid. These issues must be faced or forget all the "...abilities" we are suppose to see with objects.

The focus cannot be the code, but the temptation is stronger than ever to make it so. We now know that we cannot toss a language and development environment at our technical people and tell them to "wake us when we're object-oriented." It just won't happen!

> ***Too many development managers are unwilling to allow their people to become proficient in a new technology unless the technology promises a short-term impact on the bottom line***

For the technology to be effective, the practitioners have to learn how to wield it. That takes time. Too many development managers are unwilling to allow their people to become proficient in a new technology unless the technology promises a short-term impact on the bottom line. This "quick and dirty" approach tends to produce a collection of unrelated single-purpose tools, a hodgepodge of technology and no long-term solution. In fact, if you compute the cost of this nonintegrateable, frequently redundant "tool set," you generally find that the bottom line did not improve.

PAINFUL SOLUTION

The solution to this problem is even more painful. First, the expensive toy solutions must be disposed of. The disposal process itself is an expensive proposition. Next, a well-planned, unified technology implementation, including both technical and nontechnical aspects, must be undertaken. Often, this solution is the one that was rejected for the short-term bottom line solutions;it probably costs more now than it would have back then; and the developers still have to go through the start-up process. There are no free lunches.

Many of us who were developers ten years ago are now managers. We have already seen cultural, social, and political mistakes made in implementing structured techniques, CASE, code generators, and other new technologies. Hopefully, we have learned from those mistakes and will not obstruct those now trying to implement objects. Surely, our experience will enable us to support developers learning how to grow changeable, extensible, maintainable, and reusable code. Maybe when Rip Van Winkle awakens we will be object-oriented. Won't he be surprised! ≡≡

ENVIRONMENTS≡≡

Visual programming object language environments — a revolutionary approach

Sanford B. Klausner

Limiting expression of the multiple-dimension program world to a one-dimension textual representation severely limits our ability to comprehend and express complex computational ideas.

> *Throughout the history of the computer industry, programming languages have been primarily expressed in text. This practice has limited the utilization of the human mind to think analytically, logically, and verbally.*

CURRENT TECHNOLOGY PROBLEMS

Three important trends are leading to the creation of new software environments in which the developer employs pictures rather than text alone to design and "code" programs:

1. The task of software creation remains too difficult and expensive using text-based programming languages.
2. Users need a simple, expressive, interactive tool they can use to tailor complex applications to their needs.
3. Programs must be portable across an increasing variety of user environments.

Traditional CASE

The use of graphics in the software development process is not new. Graphic representations have addressed the design stage with CASE tools based upon functional decomposition and dataflow methodologies since the late 1970s. These "top-down" design strategies have numerous limitations and do not map well into many problem domains. While graphics have been used extensively in the design stage, the programming stage has remained textual. Attempts to automatically create code from the design have been limited to specific domains. Thus, a significant barrier remains in the software lifecycle using traditional CASE.

Object-oriented CASE

Object-oriented software development is generally a "bottom-up" design strategy based upon reuse and refinement of existing classes. Object-oriented developers are recognizing the value of front-end graphical design tools. Attempts to create graphical views of object-oriented languages have been limited due to the general mismatch between the graphical design and textual programming realms.

Software views

Problems solved in an object model are void of the rigid top-down calling tree structure associated with the functional decomposition model. The primary advantages of the object model is the way it can handle evolution, promote reusability, and model natural systems. While the lack of a rigid tree structure provides these advantages, it creates a new problem of comprehending the static and dynamic relationships of a system's objects. Current object-oriented language development environments lack tools to comprehend these complex system relationships.

Source code support

Object-oriented source code is *more* complex than procedural source code due to required extensions to support the class/object notion. These extensions have led to a serious overloading of the textual languages. This overloading complicates the task of comprehending source code, which directly impacts the reuse advantage of the object-oriented approach. Except for generic classes, most reusable code requires refinement of underlying instance attributes and

Sanford B. Klausner, inventor, founder, and president of V4th Company, has pioneered the commercial development of visual programming technology. He founded Automated Intelligence Corp. and Bio-Electronics, Inc., where he developed a wide range of technologies, including interactive voice response and building automation systems along with audiometric, EEG, and EMG biomedical instruments. Klausner can be reached by phone: (408) 374-1409, Fax: (408) 866-4502, Compuserve: 72560,1054, or mail:V4th Company, 2290 Valerie Ct., Campbell, CA 95008.

methods to meet application-specific requirements. At this level, reusable code suffers from the same inherent support difficulties that have plagued the industry throughout its history.

Graphical user interfaces

NextStep exemplifies a market trend of providing an interface builder for the development of complex graphical user interfaces. This trend can also be observed in numerous GUI tools used to develop X/Window, Open Look, and PM graphical widgets. These tools are limited to the creation of common window components while textual languages are used to complete applications. Thus, these tools present a second mismatch between the graphical and textual realms. The potential for graphical representation goes way beyond scroll bars and radio buttons which current GUI tools do not support without dropping back into a textual language.

Language/database integration

The use of object-oriented concepts in applications that generate and manage large number of objects and complex relationships among them has given rise to database systems that directly capture the object semantics. Attempts to develop SQL extensions and declarative query interfaces to the relational model have not been able to capture the rich semantics of the object model and remain far too complex for the intended user population. Also, attempts to develop an effective interface between graphical database scheme and textual database language represents a third mismatch.

Language environments

While current object-oriented programming languages are a true advance over structured descendants of FORTRAN, such as C, Pascal, and Ada, they all miss the crucial design element of using the object concept on their own development environment architectures. It is now feasible to implement sophisticated *visual programming object language environments* by integrating contemporary technologies using a new architectural approach.

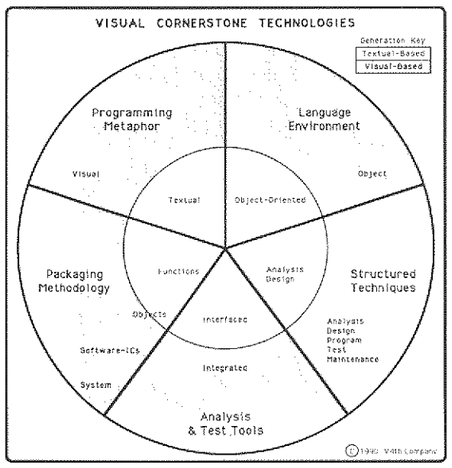


Figure 1.

VISUAL TECHNOLOGY SOLUTIONS

Visual cornerstone technologies

Visual programming metaphors (see Figure 1) represent an exploitation of our nonverbal capabilities. Textual representations can be augmented with graphics and *colors* to capture the essential syntax and semantic properties of the multiple-dimension program world. Unloading text into these alternative representations increases program comprehension with explicit pictures that represent multiple software structural views.

An **object language environment** is based upon abstracted generic operational glyphs,* which comprise basic computational concepts. Each glyph is a formal object with attribute and behavior properties. An object language architecture allows each glyph to contain message linkage to other program components providing automatic syntax, self-test, and other related environment functionality. Software **packaging methodology** in a visual environment can express the function, object, and Software-IC® levels in distinct but complementary graphical metaphors. A Software-IC is a collection of objects that perform a complex set of operations. A set of Software-ICs can be connected to form a complete system application.

Structured techniques provide a generic formalism for full lifecycle software support from analysis through testing and maintenance confined within the object model. A visual programming object language represents a homogeneous environment that effectively removes the software lifecycle barrier by *representing the design and program in one uniform graphical metaphor*. Captured ideas naturally evolve into a concrete design and early prototype. A prototype naturally evolves into efficient production software. All software components are maintained as persistent objects that can freely evolve over time, greatly reducing long-term maintenance expense.

The high degree of cohesion between an object language and an underlying object database enables fine-tuning of the run-time characteristics of an application

* The semantics of a language is defined as the meaning or relationship of meanings of its glyphs. A glyph is a symbol that conveys information. A visual programming language consists of a set of glyphs and a semantic model to give meaning to expressions made up of glyphs.

®Software-IC is a trademark of Stepstone Corp.

Code focus

There was little perception that structured techniques changed the development lifecycle. The emphasis was still on writing code, not on analysis and design. The echo still ringing through the halls of development departments everywhere was, "When will the coding start?"

Practice

There is an old line: "How do you get to Carnegie Hall?" The answer, of course, is *"Practice."* To get good at anything, you must practice. We needed practice using the technology; practice at working in teams; practice at articulating details of the application accurately and precisely; practice with a new day-to-day work routine. We needed practice, but there was no time to practice. "When will the coding start?"

Experienced support

We missed experienced support to get us over "sticky" technical details quickly. Though there was a group of people responsible for supporting projects using structured methods, these so-called experts were not. They had been through the same training as we had, but had never developed using the techniques. They were as confused as we were; and less experienced. They regurgitated theory; we needed real hands-on expertise. To make matters worse, no one dreamed that we needed support on nontechnical and management issues.

Learning teamwork

This industry attracts loners. For the most part, we had never really worked in teams before. Even though we referred to project teams, there was little interoperability among "team" members. The new technology couldn't work without team cooperation. Though we thought learning the new technology was sufficient, learning to work together effectively was perhaps the more important lesson.

User involvement

The organization tried to sneak the new technology in, never involving the users. To the users the delays were just another manifestation of development organization ineptitude. They were never aware that the new technology could save them time and money if they made an investment up front. That up-front investment was in the form of start-up costs, learning and practice. By attempting to rectify the below-standard image, the organization reconfirmed the users' prejudice.

These were primary among the issues that kept manager expectations and developer realities out of phase. Seamless, painless, invisible installation was the expectation. Reality? Well... "When will the coding start?"

So what does this have to do with object-oriented technology? Objects are reiterating the promise of "structured stuff," and then some. Hopefully, we can learned from that costly experience. New technology does not install itself. It doesn't just magically appear. Its success depends on deliberate and conscious cultivation. The names and technology are new. The management issues surrounding the implementation are not.

> ***Seamless, painless, invisible installation [of the new technology] was the expectation. The reality? Well ...***

Implementing a new technology cannot be the exclusive domain and responsibility of developers. They do not have the skill set or power to address the varied activities necessary for success. In fact, new technology can almost be thought of as being imposed on developers. Change disrupts a fairly stable, (hopefully) productive routine. Object orientation is a new team sport, a change. For successful implementation, migration, and growth, cooperation across technical and nontechnical fronts is imperative.

Senior management tends to have a "don't disturb my oblivion" attitude. They are generally standoffish and unwilling to support developers and even development managers. Quite often this lack of support is a function of fear of the technology beast. If they knew how much this fear cost, steps could be taken to rectify the situation. In fact, the situation is changing, a bit. But until recently (and still in many environments), the attitude has been unsupportive almost to the point of sabotage.

A manager's job is to provide an environment in which the technician can learn and develop proficiency in the technology, and then tune the technology to the A-natural of the organization — in short, to provide an environment in which the technology can grow. But, how is this accomplished with senior management, budgets, and deadlines breathing down our managerial necks? Unfortunately, most of the issues are nontechnical, and most of the answers are "it all depends."

ON MANAGING≡≡

Déjà Vu

Marie A. Lenzi

One of my hot buttons is the management and organizational issues enveloping the implementation of new technologies. My own experience as a developer with the introduction of a fundamentally new technology has focused me on this issue.

Marie A. Lenzi is a principle of Syrinx Corporation (specialists in object-oriented technology, development, training, and consulting) and Editor of THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY. *She has extensive experience in systems development, methodology and CASE; her work emphasizes implementation and use of methodology and CASE in professional systems environments, and training and speaking on leading edge technology. Lenzi can be reached at (212)945-9447, or by writing to 225 Rector Place, No. 12N, New York, NY 10280.*

In the early 80s, the hot new technology was structured systems development. I was developing large applications for a US multinational corporation, and my client decided to become structured. We were sent off to two weeks of "Yourdonization" and another week of methodologizing. When the training was complete, management suggested that we "wake them when we were structured." We had all learned new operating systems, languages, and tools before. Structured development was viewed as yet another new set of tools. We shrugged and said OK. But as it turned out, this was not just another new set of tools — structured development was a new way of thinking and working. Business was not business as usual. Problems began immediately.

THE REAL WORLD

Actual implementation was unlike our classroom experience. The expert instructor was not there to boost us over the uncertainties. Changes in work practices and mindset, learning curve, lack of experienced support, and so forth delayed the schedule. We scrambled. We implemented.

The applications did get installed, and the software implementation was (amazingly) considered a success — and in a sense it was. The applications were delivered on spec and in the amount of time they should (?) have taken, and they were structured (a little). But, we all knew that the implementation of structured techniques was less than stellar.

We developers felt an immense sense of failure. Other organizations had been successful at implementing this new technology (or so we thought). Why not us? There had to be a technical answer. The thinking turned to "maybe it's us, maybe we're not smart enough." Everyone else got it perfect the first time (or so we were lead to believe). Frustration and depression set in; productivity declined.

In fact, very few got it right the first time, let alone perfect; and after upwards of fifteen years people are still struggling with structured methods. Certainly, there were individual successes, but nothing in the way of a large-scale corporate return. And as far as I know, management is still waiting for that wakeup call.

Today we know that our problems were not technical. The technical issues were relatively easy to overcome: learn the technology, then develop expertise through practice. At the time, the problems were almost exclusively based in nontechnical considerations. These nontechnical issues ultimately determined the degree of success (not) achieved.

WHAT DID WE GET OURSELVES INTO?

In the early 80s, we did not truly understand the magnitude of what we were getting into. No one did, neither the technicians, nor the managers. Not even the creators and vendors of the technology fully understood the change they were bringing to the organization. Our day-to-day routine was radically altered. The impact of the change imposed on those involved directly (developers, first- and second-line managers) and indirectly (users and senior managers) was not understood at all.

to be more easily performed. Fine-tuning characteristics include virtual memory management and persistent object storage strategies.

A visual environment mirrors the intrinsic fluid nature of software by allowing design changes at any time during the application development process while maintaining consistent and uniform documentation.

Integrated analysis and test tools are much easier to include within the architecture of a visual environment using advanced hypertext and test technologies. An integrated hypertext database enables users to create document-based information pools of unlimited size, and provides users with the ability to easily discover and track relationships between individual documents. Hypertext provides a powerful core technology for analyzing, developing, and documenting software. Basic software testing is covered by formal assertions, two-dimensional path coverage, trace timing, pathway cycle spread spectrum analysis, and boundary checking. These basic testing techniques can be augmented with more sophisticated tools as experience is gained with the new paradigm.

Object communication visual modeling

The structural relationships between objects can become quite complex as objects reference each other and pass multiple messages, each containing different arguments with some requiring return values. A visual environment can provide a specific modeling tool to view the static and dynamic structural relationships of a set of objects. A set of objects from a Software-IC domain can be placed into a subject view. Subject view layers can be nested providing the ability to elegantly package very large and complex systems when programming in-the-large. An object can be placed into several subject views providing multiple system perspectives. Object communication visual modeling does not affect system behavior and is used during development, debugging, and maintenance stages for the creation and support of more effective software designs.

Uniform visual "code" representation

Traditional textual source code is replaced with two-dimensional color diagrams constructed from a small set of fundamental glyphs. A user concatenates glyphs much like "mental Tinkertoys" to solve complex problems in a highly visible, standard framework. Each glyph type has several forms of use. A fundamental glyph library provides a rich set of efficient low-level operations. This fundamental glyph approach addresses a major reason today's programs are difficult to develop and maintain because of the lack of low-level statement standardization. This thinking style unburdens a developer's mind, allowing more concentration on the problem-domain for the production of more creative, better organized object-based programs.

Next step beyond in GUI tools

Using two-dimensional drawing tools enables unrestricted expression without the limitation of predefined graphical widgets. A system expressed in graphics can be easily shared and refined. Once an abstract design becomes concrete, a visual environment provides the capability to decompose static images into their dynamic components, which become graphical program components through a single transformation.

The tight cohesion between the graphical realm and the object language semantics enable drawings to directly represent programming concepts. Constraint-based graphical programming based upon drawing elements can be easily constructed for specific problem domains. This visual construction capability allows a developer to analyze problems in new and powerful ways.

> ***A visual programming object language environment represents the fusion of five cornerstone technologies integrated into a revolutionary development medium.***

Unified database programming language

A visual environment can provide facilities for designing aggregate object schema in addition to query and data manipulation methods in a uniform metaphor. These facilities are an extension of the visual programming object language providing a unified *database programming language*. A visual programmatic interface to an integrated object database provides a powerful and easy to use tool for both programmers and end users to construct sophisticated database applications. This general-purpose interface can be extended with domain-specific declarative interfaces including forms, menus, and graphs.

CONCLUSION

The evolutionary approach to improving software productivity has not demonstrated the necessary gains required to meet the demand for new applications. A visual programming object language environment represents the fusion of five cornerstone technologies integrated into a revolutionary and powerful development medium. This power is derived by redefining basic notions of language design and represents the enabling technology to meet future software demands. ≡≡

OBJECTIVELY THINKING≡≡

Hype: necessarily (or a necessary) evil?

Jim Adams

Hype — we're surrounded by it;
we're sick of it.
We all wish it would stop,
yet none of use can give a really good,
objective definition of hype.
What is hype?

Hype is a pejorative name for advertising. To paraphrase Bertrand Russell: *"We promote, you advertise, but they hype."* Although in our more candid moments,we all may admit to using hype from time to time, hype is a charge that, generally, we lay at the door of others.

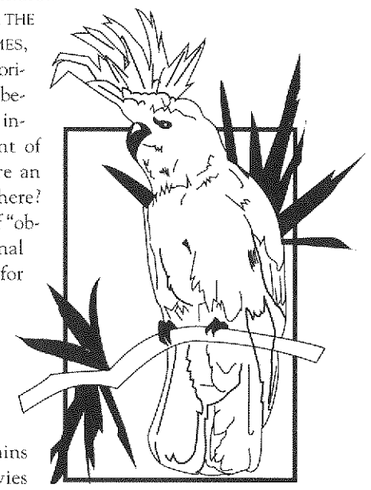
Hype invariably concerns something new. Nobody makes a big deal today about zippers, linoleum, kerosene, or dry ice, but all these once-trademarked items were hyped in their day because they were new. The same was true for "structured programming" and"top-down design" a few years ago.

Hype is like bad art: everybody knows it when they see it, but nobody is willing to say exactly what it is. This is closely related to a school of thought that considers all advertising, marketing, and public relations to be an elaborate fraud designed to plunder the consuming public. (Of course, the opposite side of this "advertising-is-bad" coin is the plight of the consumer in countries like the Soviet Union, where the utter lack of advertising forces consumers to trudge daily from store to store—often in vain — in search of ordinary items.) It's time to strip hype of its negative emotional baggage and call it what it is: advertising for something new.

Hype has been part and parcel of the computer and electronics industry since Day One. Remember when it was a big deal for a radio to be "transistorized," or for a stereo component to be labeled "solid state" as vacuum tubes were phased out? Computers have always been bigger and faster; this year's model was always better than last year's.

As microcomputers proliferated, so did the hype. A few years ago, the Next Big Step in software was the integrated application that combined a spreadsheet, a database, a word processor, and perhaps a telecommunications program in one package. Despite the hype and the predictions of industry pundits, these failed to take the market by storm. Remember IBM's Topview? Apple's Lisa? And what about OS/2, which was indifferently received despite massive hype (and is only slowly growing in use today)? Most dramatic, perhaps, has been the recent saga of artificial intelligence, where, after many years of research, its proponents emerged from academies and labs, invested millions in developing the technology, and then hyped the technology to death.

Today, a number of people have noted that "object-oriented" is fast becoming *the* buzzword for software marketers. Indeed, Andrew Pollack, writing in THE NEW YORK TIMES, notes that "object-oriented has recently become the software industry's equivalent of oat bran." Is there an ominous parallel here? Does the hyping of "object-oriented" signal hard times ahead for OOP? In general, I don't think so.



Let's face it: today, anything that works once is aped repeatedly, which explains why we have movies with titles like "Superman IV" and "Rocky V." When a promising technology such as OOP emerges from academic and research environments, it's not surprising to see a rush to label every and anything as object-oriented, leaning on the flimsiest of explanations to justify the usage.

However, it is not the use of hype that leads to adverse sales. To arrive at such a conclusion is to engage in *post hoc* reasoning. Just because failure follows the hype, it does not follow that failure must have been caused by the hype. We know that hype (since it is something everyone else does) will always be part of the equation. What will determine the overall successor failure of a technology is its ultimate ability to make things better for the consumer of the technology. For the computer industry, this means solving users' problems.

Let's take a look at what happened with AI. At one extreme, there was a herd of AI programs that were riding the wave of hype, yet that had nothing substantive to offer in terms of AI. I still recall one program that did nothing particularly intelligent (artificial or otherwise), yet that presented the user with the message "loading artificial intelligence" as it was being started. The market saw through such humbug, and this type of program had a relatively short life span. Self-proclaimed "object-oriented" programs that are nothing of the sort will die similar, ignominious deaths.

However, at the other extreme, where you had programs that very definitely did "do AI," the typical marketing effort attempted to overpower potential users with the technology and its buzzwords. Backward chaining. Frame-based representations. Rule-based reasoning. The proponents of these programs failed because they pushed the technology as an end in itself, and not as a solution to the problems of intended users. For a while, users were willing to go along with the program, dazzled by the promise of performance, until projects began to fail, and fails spectacularly. As a result, the label "AI" became the kiss of death for any application seeking to survive in the market.

So what lesson can object-oriented programming learn from this? Exactly this: Don't get hung up on proving the perfection of the technology. Don't try to dazzle prospective clients with the purity of your implementation. In the final analysis, if a product that claims to use OOP techniques doesn't address a pressing problem satisfactorily, then it will eventually fall by the wayside. All the hype in the world won't keep it afloat. If enough such products fail, then the technology fails. It follows that what is important is not measuring the "object-orientedness" of a product, but its usefulness.

Take, for example, VROOMM, Borland's Virtual Runtime Object-Oriented Memory Manager. There are basically two ways to explain how it works. First, I could use the "technoweenie" approach, and get into the guts of how, exactly, it manages memory virtually in real time and in an object-oriented manner. Alternately, I could describe what it does, demonstrate how VROOMM performs, and show how it can make your programmer's life easier.

Regarding the first approach, I say: *Yech*. Who needs this? Although such an explanation might convince you of the virtualness, etc. of the VROOMM technology, who (besides another technoweenie) would care?

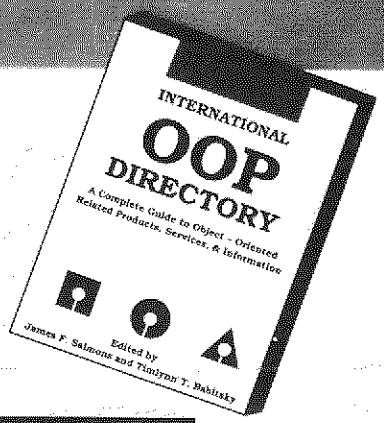
> ***I still recall one program that did nothing particularly intelligent (artificial or otherwise), yet that presented the user with the message "loading artificial intelligence" as it was being started.***

By the way, since there are serious arguments going on today regarding the "object-oriented"-ness of various things — including the C++ language — maybe the explanation would fail to convince you, and then where are we? Is the technology useless? Null and void? What if it fails to convince you, but convinces your neighbor? These and other irrelevant questions arise to muddle our view if our focus is to measure the extent of how object-oriented VROOMM is. The second approach — which shows whether the technology is worth anything — would seem to be the better one, but unfortunately, since it lacks technical depth, tends to be dismissed as hype.

What the OOP market needs is not continued fruitless sparring as to what does (or does not) constitute object-oriented programming, nor continued wailing about the profusion of hype about OOP. What the OOP market needs is the realization that bad products should simply be ignored, and that useful technology is available now for developers to use. ≡≡

Jim Adams is manager of Languages Public Relations at Borland International. His focus is on the support of Borland's Turbo Language Products and object-oriented programming technology. His varied background includes similar positions in retailing, the semiconductor industry, and advertising, radio, and television.

Go DIRECTLY TO THE SOURCE...



Introducing The

1990 International OOP Directory

The Complete Source for OOP-related Information

Unlike absolutely any other buyer's guide that's on the market today, this unique publication covers virtually every aspect of the O-O field. That means you'll find detailed information on companies, products, consultants, as well as reviews, significant published articles, books, training, conferences and inside info — in one convenient sourcebook ... **The 1990 International OOP Directory.**

The facts you locate will be accurate, complete and up-to-the-minute — providing you with reliable information needed before you make an OOP-related purchasing decision. **The Directory** is edited by highly renowned OOP industry experts, Jim Salmons and Timlynn Babitsky. Its organized format directs you quickly to the section you're looking for. Complete cross-referencing helps you locate valuable information faster and easier than ever before.

Here's What's In It for You!

Δ **Company Guide**—A-Z listing of over 200 OOP-related companies. Synopses of companies' backgrounds, contacts and products give you vital corporate information — with cross-referenced indexes by product names, platforms supported and base languages used.

Δ **Product Guide**—Nearly 300 products alphabetized by product category — with cross-referenced indexes of base languages, operating systems or platforms, product categories and prices. Includes insightful published product reviews.

Δ **Consultant's Guide**—with an alphabetical listing of contacts and specialties.

Plus an OOPs In Print Update

Δ **Selected Reprints** of landmark articles from the *Journal of Object-Oriented Programming*, *The C++ Report* as well as the *HOTLINE on Object-Oriented Technology.*

Δ **Articles** - Citation listings of published articles, cross-referenced by languages and major subject areas.

Δ **Books** — Complete listings of published books and in-print books, listed by author and by subject.

Δ **Conferences, Seminars and Training** — Alphabetized list by sponsoring organizations, with dates, courses, lengths, prices and contacts to help you plan your calendar and budgets. Includes upcoming and regularly recurring events.

The 1990 International OOP Directory Will Assist You In —

Δ Quickly finding exactly what you need.

Δ Sharpening your budget and also placing all product pricing information at your fingertips.

Δ Locating products that are compatible with your operating system.

Δ Spotting up-to-the-minute trends in the marketplace.

Δ Contacting those software manufacturers you want to do business with.

Δ Providing access to pertinent landmark articles you may have missed. Using the spectrum of published articles as your handy "clipping service."

Δ Getting product reviews that can save you time, money and inconvenience.

Δ Scanning years of costly research from important selected articles.

Δ Cutting through the clutter to determine the books you want to read.

Stop wasting your time with limited "bits and pieces" of information from more generally focused software magazines. Simply turn to the only directory that tunes you in to the entire object-oriented field. Now you can easily access OOP information whenever you need it ... directly.

This "total sourcebook to the O-O industry" is like having a well-connected consultant at your finger tips. Anytime. Full-time. The simple fact is you'll find everything that has anything to do with object-oriented technology right here. Your satisfaction is guaranteed.

Simply purchase **The 1990 International OOP Directory** at no risk. If you are not *100% satisfied* for any reason, return it for a full refund within 15 days. No questions asked.

OOP Directory Order Form

❑ Yes, please send me ___ copies of **The 1990 International OOP Directory.** $59 ISSN#1050-4354

US orders add $3 for postage & handling. Foreign orders add $10 for air mail.

❑ Check enclosed (payable to OOP Directory)

Foreign orders must be prepaid in US dollars drawn on a US bank

Order by phone **(212)274-0640** Fax: **(212)274-0646**

Charge my ❑ Visa ❑ Mastercard

Card# _____ Exp. Date _____

Signature _____

❑ Standing order (automatically send me 1991 edition and bill me)

Name _____

Phone _____ Title _____

Company _____

Address _____

City _____

State/Country _____ Zip/Postal Code _____

Send paid orders to **OOP Directory, c/o SIGS Publications, Circulation Dept., 588 Broadway, Suite 604, NY, NY 10012**

COKD

≡ Your Turn ≡

How about it? How are we doing? This is the place to send us your message. Tell us what you like and what you don't, what you would like but don't see — and whatever other feedback you'd like to send. Send comments to Marie A. Lenzi, 225 Rector Place, NY, NY 10280; fax to THE HOTLINE, *(212)274-0646; or, if you're ordering, send along with your order.*

Please check whichever box applies:

	Read entirely	Scanned	Found helpful	Will refer to	Didn't read
Feature	❑	❑	❑	❑	❑
Databases	❑	❑	❑	❑	❑
Project Management	❑	❑	❑	❑	❑
Down Under	❑	❑	❑	❑	❑
Book Review	❑	❑	❑	❑	❑
FYI	❑	❑	❑	❑	❑
Product News	❑	❑	❑	❑	❑
Calendar	❑	❑	❑	❑	❑

Please rate the following **1** (dislike or strongly disagree) to **10** (like or strongly agree):

Presentation of material __ Comments: _____

Accuracy of material __ Comments: _____

Overall helpfulness in your job __ Comments: _____

What else would you like to see in THE HOTLINE? _____

Any other comments? _____

_____

_____

Name, title, phone (optional): _____

hotline on OBJECT-ORIENTED *technology*

❑ ***Yes,*** plug me into the latest thinking and developments in object-oriented technology. Enter me as a subscriber at the term marked below and rush me the current issue. This is a risk-free offer — I may cancel my subscription at any time and promptly receive a refund for the unused portion.

1 year (12 issues) ❑ $249

2 years (24 issues) ❑ $478 *(save $20)*

Back issues @ $25 each *($28 foreign) — see Page 23:*
Vol.1, Nos. _____

(outside US, add $30 per year for air service)

❑ Phone order
Call Subscriber Services
(212)274-0640

❑ Fax order
Fill out term and name/address information, then fax to:
(212)274-0646

❑ Send me a copy of
THE INTERNATIONAL OOP DIRECTORY
@ $62 ($69 foreign)
(all directory orders must be prepaid; fax credit card information or mail to SIGS, 588 Broadway, #604, NYC 10012; make check payable to OOP Directory. Foreign orders must be in US dollars drawn on US bank.)

D0LD

❑ Bill me
❑ Check enclosed
❑ Credit card orders
❑ MasterCard
❑ Visa

Mail (and make out check, if applicable) to:
THE HOTLINE
Subscriber Services
P.O. Box 3000, Dept. HOT
Denville, NJ 07834
(foreign orders must be prepaid in US dollars drawn on a US bank)

Card# Expiration Date

Signature

Name

Company

Street/Building#

City/Province

ST/Zip/Country

Telephone

D0LD

Send me a complimentary copy of your related publication(s):

❑ THE JOURNAL OF OBJECT-ORIENTED PROGRAMMING

❑ THE C++ REPORT

Call Subscriber Service
(212)274-0640
or send to
SIGS, 588 Broadway #604
New York, NY 10012

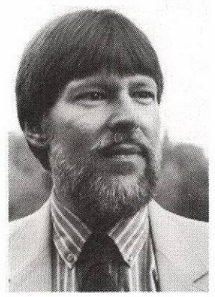
D0LD

hotline on OBJECT-ORIENTED *technology*

VOLUME 2, NUMBER 1 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES NOVEMBER 1990

Objects by teamwork

One of the most promising trends in software methodology (outside of object orientation, of course) is a rising interest in the "peopleware" issues of software development, especially in how to build and manage more effective project teams. (Thomsett, 1990; Zahniser, 1990). The use of object-oriented development rather than traditional function-oriented methods can have a profound impact on how work is organized and managed over the project lifecycle. Effective teamwork may offer one of the best ways to achieve more of the full promise of building systems with enhanced reusability and more robust modeling of applications, especially for larger systems. Teamwork models can help carry object-oriented technology beyond its roots in experimental prototyping and the isolated work of gifted programmers into the world of large applications in banking, telecommunications, aerospace, and other areas.



Larry L. Constantine

Teamwork models offer solutions to a number of key issues in object-oriented development, including:

1. maintaining access to applications domain expertise,
2. effective management and utilization of reuse libraries, and
3. development of framework software.

Before returning to these issues, some general aspects of software teamwork will be explored.

WHAT'S A TEAM?

Perhaps the hardest thing for many developers and their managers to understand is the difference between a real team and the more common project group, little more than a random grouping of available programmers who are assigned to a project and told by their manager, "Now you're a team."

A team that actually functions by teamwork achieves a synergy that makes what they can accomplish add up to more than the sum of their individual efforts. The random programming horde, as they step over each other and on each

continued on page 3 ...

IN THIS ISSUE

FEATURE — Objects by teamwork *Larry L. Constantine*	1	2	FROM THE EDITOR
DATABASES — Race for the database pennant: a third-generation manifesto *Dr. Jacob Stein*	7	19	FYI — Industry publication excerpts
PROJECT MANAGEMENT — Talking to project managers — project team job descriptions *Adele Goldberg & Kenny Rubin*	12	22	PRODUCT ANNOUNCEMENTS
DOWN UNDER — The Australian object-oriented scene *Dr. B. Henderson-Sellers*	17	23	CALENDAR
BOOK REVIEW — "Designing Object-Oriented Software" *Reviewed by Douglas W. Bennett*	18	24	YOUR TURN — Reader feedback

FROM THE EDITOR ≡≡

Object-oriented methods can have quite an impact on how development projects are organized and managed. **Larry Constantine** presents various teamwork models that may offer one of the best way to achieve more of the promise of the object "...ability."

Another skirmish in the hotly contested "database wars" may be underway. This time, however, it is not among the OODBMS developers/vendors, but between the object and relational database people. The relational proponents have issued their own Third-Generation Database System Manifesto. **Jacob Stein** compares and contrasts the technical aspects of this document to the Object-Oriented Database Systems Manifesto published in December 1989. Tim, Mary, Drew, Tom, David, any comments?

After a summer hiatus, **Adele Goldberg** and **Kenny Rubin** return to continue "Talking to Project Managers." In the September editorial, I offered a broad discussion of new job functions created by the advent of object technology. This month, they take a detailed look at project team job descriptions.

Dr. Brian Henderson-Sellers of the University of New South Wales continues to keep us current on The Australian Object-Oriented Scene.

Because object technology is more a function of design than implementation, analysis and design are getting more serious attention than ever before. Object analysis/design seminars, classes, tutorials, presentations and pontifications abound. Many theorists and practitioners are trying to catalogue the animal. And even though they don't agree on specific approaches, each articulates the need for continuous access to application expertise, emphasizes the need for precise requirements definition, and differentiates between analysis notation and analytical skills.

The iterative development model (as opposed to a waterfall model) clearly shows that analysis decisions may need to be made far into implementation. It is well understood that guesses, educated or otherwise, are not advisable. The domain expert has a significant contribution to make throughout the lifecycle, not just in the early and late phases, as in other lifecycle models.

Frequently, when a problem is difficult to solve, it has been ill-defined. Once a problem is precisely defined, solutions can be identified. However, requirements specifications (problem definition) may be the most difficult and illusive part of the development process.

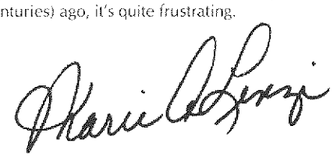
Among the skills of a good analyst is the ability to interview, listen and hear; to read between the lines; to identify omissions and evaluate apparent irrelevancies; and to communicate this understanding to both the user and developer community in a language each understands. An experienced analyst recognizes the trade-offs between the user's needs and the available implementation technology.

The proper skill set and level of experience for an analyst has long been evaded. The job is too often improperly awarded. Frequently, the interpreter has been a fine developer or designer who has been elevated, a university student on break, or a newly graduated student with low technical and professional experience. Not fair to anyone involved!

But help appears to be on the way. We are talking about analytical skills, a lot. We are beginning to accept the fact that effective interaction with the user, coupled with well-honed interpersonal skills and technical knowledge, are key to accurate and precise requirements specification. But none of this is news; we've known this for a long time.

The difference now is that those who previously spoke softly on the topic, the engineers and technicians, are beginning to recognize how critical these skills are and to speak out in support of them. At last steps are being taken; but only first steps. Before we are steady on our feet, we have some work to do.

But let's not redundantly reinvent. The problem has already been addressed and solved in other engineering and social disciplines. What we need to do is adapt and refine. Discovery is expensive, and when you learn that the problem was solved decades (even centuries) ago, it's quite frustrating.



Advisory Board

Jim Anderson, *Digitalk, Inc.*
Grady Booch, *Rational*
Larry Constantine, *Consultant*
Adele Goldberg, *ParcPlace Systems*
Mary E.S. Loomis, *Versant Object Technology*
Bertrand Meyer, *Interactive Software Engineering*
Reed Phillips, *Knowledge Systems Corp.*
Trygve Reenskaug, *Taskon A/S*
Bjarne Stroustrup, *AT&T Bell Labs*
Dave Thomas, *Object Technology Int'l*
Zack Urlocker, *Borland International*
Steven Weiss, *Wayland Systems*

Erratum: Doug Bennett's phone number was incorrectly given in the September issue. The correct phone number is (203)426-9511.

Editor
Marie A. Lenzi

SIGS Publications, Inc.
Richard P. Friedman, Group Publisher
Margherita Monck, General Manager
Elisa Varian, Production Manager
Susan Culligan, Art Director
Melanie Younossi, Marketing Manager
Gregory Tsichlis, Circulation Manager

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012 (212)274-0640.

© Copyright 1990 SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the U.S. Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publisher.
Mailed First Class. Subscription rate—1 year (12 issues) $249, Foreign and Canada $279. Single copy $25.

POSTMASTER: Send address changes & subscription orders to THE HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville NJ 07834.

Submit editorial correspondence to Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280. (212)945-9447.

hotline on OBJECT-ORIENTED technology

Back issues available

Vol.1, No.12/Oct. '90 ≡ Crown jewels: the value of OOT in MIS departments • *Jeffrey Bonar* ≡ DATABASES: Why is an object database faster than a relational system? • *Mary E.S. Loomis* ≡ ENVIRONMENTS: Visual programming object language environments • *Sanford B. Klausner* ≡ OBJECTIVELY THINKING: Hype — necessarily (or a necessary) evil? • *Jim Adams* ≡ ON MANAGING: Déjà Vu • *Marie A. Lenzi*

Vol.1, No.11/Sep. '90 ≡ Introducing OOPS through design — it's all in the words • *Seth Copen Goldstein* ≡ DOWN UNDER: The Australian object-oriented scene • *Dr. B. Henderson-Sellers* ≡ Objects on tour (Review of Borland's OOP World Tour) • *Marie A. Lenzi* ≡ ANALYSIS & DESIGN: Design principles for modular product development • *Douglas W. Bennett* ≡ MANAGEMENT: Asset management and object-oriented technology • *Trevor Hopkins & Brian Warboys*

Vol.1, No.10/Aug. '90 ≡ WyCASH+: An application built within an OOP environment • *Paul McCullough & Nancy Deshler* ≡ DATABASES: Mathematics and methodology • *Robert R. Martin* ≡ PROJECT MANAGEMENT: Talking to project managers: Organizing for reuse • *Adele Goldberg & Kenny Rubin* ≡ STANDARDS: CALS Expo–89 Conference • *Alan Purdy*

Vol.1, No.9/July '90 ≡ Cognos and Eiffel: a cautionary tale • *Burton Leathers* ≡ What's in an object anyway? • *Andrew J. Novobilski* ≡ LANGUAGES & ENVIRONMENTS: The impact of Windows 3.0 on the OOP market • *Zack Urlocker* ≡ PROJECT MANAGEMENT: Talking to project managers: Introducing reuse into the development process • *Adele Goldberg & Kenny Rubin*

Vol.1, No.8/June '90 ≡ A taste of the USENIX C++ Conference • *Mary H. Wells* ≡ DATABASES: Comparing relational and object database management systems part 2 — Composites, flexibility, and extensibility • *Robert R. Martin* ≡ PROJECT MANAGEMENT: Talking to project managers: Case studies in prototyping, part 3 • *Adele Goldberg & Kenny Rubin* ≡ ANALYSIS & DESIGN: Object-oriented requirements analysis • *Edward V. Berard*

Vol.1, No.7/May '90 ≡ Object-oriented development: not business as usual • *Mark Lorenz* ≡ DATABASES: Comparing relational and object database management systems part 1 — Single paradigm programming is the key difference • *Robert R. Martin* ≡ PROJECT MANAGEMENT: Talking to project managers: Case studies in prototyping, part 2 • *Adele Goldberg & Kenny Rubin* ≡ ANALYSIS & DESIGN: Understanding the recursive/parallel lifecycle • *Edward V. Berard*

Vol.1, No.6/April '90 ≡ Lifecycle approaches • *Edward V. Berard* ≡ DATABASES: A data manipulation language for C++ developers — a proposal • *Thomas Atwood* ≡ PROJECT MANAGEMENT: Case studies in prototyping, part 1 • *Adele Goldberg & Kenny Rubin* ≡ ANALYSIS & DESIGN: I already learned in kindergarten all I needed to know about object-oriented design • *Niranjan Ramakrishnan*

Vol.1, No.5/March '90 ≡ Objects of desire • *Christopher M. Stone* ≡ DATABASES: OMG: The state of the object-oriented DBMS market • *Tom Atwood* ≡ PROJECT MANAGEMENT: Talking to project managers.... • *Adele Goldberg & Kenny Rubin* ≡ INTERNATIONAL BEAT: The Australian object-oriented scene • *Dr. B. Henderson-Sellers*

Vol.1, No.4/February '90 ≡ Greed: a complete example • *Tom Love* ≡ DATABASES: OMG: an industry push for standards • *Tom Atwood* ≡ PROJECT MANAGEMENT: Talking to project managers.... • *Adele Goldberg & Kenny Rubin* ≡ LANGUAGES & ENVIRONMENTS: OOP in the 1990s • Zack Urlocker

Vol.1, No.3/January '90 ≡ Adopting object-oriented techniques: management and organizational impacts • *Marie A. Lenzi* ≡ DATABASES: The object-oriented database system manifesto: a consensus from academia • *Tom Atwood* ≡ LANGUAGES & ENVIRONMENTS: Which language to choose, revisited • *Jeff McKenna*; On language wars • *Bjarne Stroustrup*

Vol.1, No.2/December '89 ≡ OOPSLA '89 • *Jeff McKenna* ≡ GUEST ARTICLE: C'ing into the future • *Keith W. Howard* ≡ DATABASES: Standards for OODBMS: a manager's guide • *Tom Atwood* ≡ ANALYSIS & DESIGN: Synthesis: an object-oriented analysis and design method • *Meilir Page-Jones & Steven Weiss* ≡ LANGUAGES & ENVIRONMENTS: From languages to environments • *Zack Urlocker*

Vol.1, No.1/November '89 ≡ Which object-oriented language should we choose? • *Ken Auer* ≡ DATABASES: An introduction to object-oriented database management systems • *Tom Atwood* ≡ ANALYSIS & DESIGN: Understanding object-oriented analysis • *Sally Shlaer & Stephen J. Mellor* ≡ LANGUAGES & ENVIRONMENTS: Object-oriented programming: myths and realities • *Zack Urlocker*

To order back issues, use the order coupon on back page

Reprints of individual articles available — call Kathy McKendrick at Reprint Resources (215)643-9140

Calendar

THE HOTLINE CALENDAR presents conferences and meetings that focus exclusively on object-oriented technology. To have a meeting or conference listed, please send the dates, conference name and location, sponsor(s) and contact name and telephone number to the Editor: Marie A. Lenzi, 225 Rector Place, Suite 12N, New York, NY 10280.

November 6–9	November 8	November 26–28	November 28–30	December 5–6
Software Development — Fall Boston, MA Sponsor: Miller & Freeman Conferences Contact Angela Hoyt (415)267-7649	**O-O Structural Design** Washington, DC Sponsor: Digital Consulting, Inc. Contact Sue Howard (508)470-3870	**O-O Systems Symposium** Madrid, Spain Sponsor: Digital Consulting, Inc. Contact Sue Howard (508)470-3870	**TOOLS-'90 Pacific** Sydney, Australia Sponsor: Interactive Software Engineering. Contact Myriam Wever Phone (+61 2) 810 2502 Fax (+61 2) 810 3726	**O-O Systems Symposium** Chicago, IL Sponsor: Digital Consulting, Inc. Contact Sue Howard (508)470-3870

Product Announcements

Product Announcements is a service to the readers of THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY; it is neither a recommendation nor an endorsement of any product discussed.

Choreographer

GUIdance Technologies
800 Venial Street
Pittsburgh, PA 15212
412 231-1300

GUIdance Technologies has released version 1.2 of **Choreographer**, a software tool for developing graphic user interfaces (GUIs) for OS/2 Presentation Manager. Choreographer is particularly effective in creating transaction-intensive applications such as airline reservation and invoice tracking systems. Version 1.2 enhancements include:

- Interactive Interface Editor: provides a complete set of standard PM and CUA layout tools for interface development, enabling programmers to lay out and automatically link code to all standard interface components; provides for the creation of tab circuits, field validation, accelerator keys and resizing specifications.
- Development Module Editor: enhances team programming productivity by providing a centralized management file for program components.
- C structures and COBOL records: can share C structures and COBOL records with other application programs. No copying or special treatment is required.
- SQL Client: allows developers to use SQL standard for database access, enabling them to create queries and send and receive data without knowledge of SQL database APIs.
- 3270 and 5250 Communications Extensions: simplifies Choreographer's ability to replace existing mainframe-based textual interfaces with no changes required to mainframe applications.
- LU 6.2–APPC Support: simplifies Choreographer's ability to use this communication protocol.
- Interactive Debugger.

Versions of Choreographer are under development for Windows and UNIX.

Draw-80

Knowledge Systems Corp.
114 MacKenan Drive
Suite 100
Cary, NC 27511-6446
919 481-4000

Draw-80 is an advanced object-oriented graphics framework that allows Smalltalk-80 users to add high-level object-oriented graphics capabilities into Smalltalk applications. It combines the programmable control of a graphics library with an intuitive graphical editing interface. Draw-80 helps Smalltalk-80 developers in the following ways:

- User gets 24 drawing tools ranging from text to isosceles triangles and from filled circular arc to parallelograms.
- Allows for the manipulation of graphical object in dozens of ways.
- Provides for the grouping and ungrouping of graphical objects.
- Delivers dynamic scaling of graphical objects.
- Provides angle rotation, alignment and pattern fill with any of 60 patterns.

In addition, users get the actual Draw-80 source code. Smalltalk-80 class/code libraries that are provided in Draw-80 can be integrated into other Smalltalk-80 applications to support dynamic graphics. Draw-80 requires the same workstation or personal computing hardware as is required for the corresponding Objectworks for Smalltalk-80 program from ParcPlace Systems. On systems that do not support virtual memory, a minimum of 4 Megs of RAM is recommended. Draw-80 is priced at $499 and is available immediately from KSC.

Microtec Research

2350 Mission College Blvd.
Santa Clara, CA 95054
800 950-5554

Intel Corporation and Microtec Research Inc. released a set of integrated software development tools for Intel's 80960 family of microprocessors. Intel assisted in the development by providing advanced information and technical cooperation to Microtec. Microtec Research based the software tools on its technology to take advantage of the advanced features of the 80960. The toolkit consists of an optimizing ANSI C compiler, a source level debugger based on Microtec Research's XRAY Debugger and an assembler with linking loader and librarian. The components of the toolkit are designer to function together as an integrated whole and provide a superset of the features found in typical development tools.

Send Product Announcements to Marie A. Lenzi, 225 Rector Place, No. 12N, New York, NY 10280. Include company name, address, and phone number.

continued from page 1

other's toes and pull at cross-purposes or work without integration, may be capable of even less than if they all simply went off on their own. The genuine article, teams that have been constituted and organized through a deliberate process of team building, have unity of purpose, interdependence in action, accountability, and functioning organization. Such a team doesn't just happen.

It requires establishing a shared history, a common culture, and an effective internal organization. Research in various industries shows that well-structured, well-supported teams always outperform mere "bunches." Given an opportunity to focus on their own structure and member roles, teams can increase productivity by 50–200% (Thomsett, 1990).

WHY TEAMS?

There are many reasons for doing software development in teams. Besides the obvious potential for building larger systems that hang together better, smooth-functioning teams have many advantages. They can draw on and integrate a greater variety of skills, talents and viewpoints; such diversity gives them extra resources and versatility. Teams are also inherently self-monitoring compared to scattered individuals or less closely working project groups. When work is done collectively and visibly, departures from standards or from specs are less likely to slip by. Effective teams also provide mutual support that can make a crucial difference in mission-critical or breakthrough projects (Straus and Spiwack, 1990). And, for many people, teams afford one of the best ways to achieve the twin goals defined by Australian consultant Rob Thomsett (1980), namely, to "have fun and do good work."

Object orientation and teamwork fit well with each other in a number of ways. The exploratory modeling and iterative prototyping approaches so commonly used in object-oriented development lend themselves to group processes where people can bounce ideas off each other and forge richer and more robust models out of their distinct viewpoints. Identification of reusable components and component collections can be facilitated by conjoint analysis and design processes. Some of the development tools, such as the brilliantly simple "CRC" cards or "class cards" used in responsibility-driven design (Wirfs-Brock, Wilkerson, and Wiener, 1990), are tailor-made for group problem solving. In my experience, these fancied-up index cards have proven to be not only an effective group learning tool, but can help teams to construct and explore object-oriented designs on "real" projects.

TEAMWORK PARADIGMS

There are many group approaches to software development (Zahniser, 1990). To make sense of the variety of teamwork models, it helps to understand the underlying assumptions behind different teamwork approaches. Just as the world of software has seen the traditional function-oriented paradigm challenged by the object-oriented world view, so teams can be understood in terms of the organizational paradigms on which they are based (Constantine, 1986, 1989, 1990).

Modern teamwork models used in object-oriented software development generally take one of two basically different tacks. The traditional tack organizes work into assigned job functions based on technical responsibilities and uses conventional notions of project management and control, in which programmers and other professionals report to first-level supervisors or group leaders who report to project managers, and onward and upward.

Such project teams are said to be based on a closed organizational paradigm, one of traditional authority and control. Teams organized by this paradigm depend on strong, ongoing direction from a manager, along with clearly defined and separated roles and responsibilities. They are controlled through top-down decision processes. The highly structured model described by Goldberg and Rubin (1990a) is an example of a closed paradigm organization adapted to object-oriented development.

Another major variant of teamwork organization being applied to software development is the open paradigm. An open paradigm team consists of collaborating peers who make decisions and coordinate their activities through group consensus building. Quite unlike the closed paradigm team, an open team may have very fluid and flexible roles with shared or rotating responsibilities. IBM's Joint Application Design (JAD) model (Wood and Silver, 1990) and the structured open team (Constantine, 1989; Rettig, 1990; Thomsett, 1990) are based, at least in part, on the open paradigm. There are other fundamentally distinct variants of organizational paradigms, at least four major themes in all (Constantine, 1989, 1990), but only these two classes — open and closed — need be considered here.

STRUCTURED OPEN TEAMS

The structured open team is a model devised specifically for technical project teams designing and building systems. Such teams have been used with good success by developers in both the United States and Australia (Thomsett, 1990). The model integrates selected aspects of closed paradigm and open paradigm organization to get more of the practical payoffs of open paradigm teams while compensating for some of the weaknesses.

Closed paradigm teams have a straightforward struc-

ture that makes them easy to set up and efficient in operation. They can perform well, with clear accountability to the larger organization, especially on straightforward "tactical" projects that do not demand great innovation. On the down side, they can be short on inventiveness, are critically dependent on good management, and do not facilitate high levels of personal ownership in the group's product. Open teams, on the other hand, are better at problem solving, producing creative yet practical solutions in complex environments where the free sharing of information is critical for success. They also tend to generate a stronger sense of ownership among team members who personally "buy into" the joint solutions arrived at by consensus. Without good communication and consensus-building skills, however, they can become mired in wheel-spinning debates and trapped by unfortunate compromises.

> ***Close, robust modeling of the underlying conceptual structure enhances reusability, extensibility, and maintainability ...***

In a structured open team, a single technical leader is held accountable to the larger project or organization, but most leadership responsibilities are considered to be the responsibility of the team as a whole, to be shared and rotated among all members. The model incorporates the notion of "cross-training" to spread skills among members. Well-established methods for meeting management (Doyle and Strauss, 1982), including the need for a neutral facilitator and a continuously maintained collective "group memory," can be adapted to the realities of software project teams. For example, instead of requiring a trained, outside facilitator as in the JAD model, structured open teams treat process facilitation as a role to be rotated among team members.

MEMORY MANAGEMENT

One of the problems with so many project teams is that, even when they produce a good product, they don't necessarily keep track of how they got there — what decisions they made along the way, how they resolved them, mistakes overcome, approaches rejected, and other aspects of the underlying software engineering. Many modern teamwork models overcome these problems through a group memory that carries the process history along with the evolving design of the group. In most approaches, an information manager, scribe, or recorder is made responsible for maintaining a visible, accessible, and complete memory of what the group has done and how they got there.

Unfortunately, this is an onerous responsibility that no one really wants, at least not for long. And it's no good assigning a secretary or hiring an outside process recorder because, for success in a programming project, the information manager has to understand what he or she is recording and cataloging. Besides, most small projects can't afford the overhead of a hired recorder/librarian. In a structured open team, responsibility for information management is rotated among regular project personnel.

MEETINGS AND MANAGEMENT

Success as a team may require a change in attitudes about meetings. Among software developers, meetings are typically viewed as "down time," an annoyance that has to be put up with to be free to do "real work." Coordinating the efforts of a team does take time, but this interaction is the very lifeblood of collaboration that brings out the best in team members. An effective team may spend as much as half its time in meetings of one sort or another and still produce more and better code than isolated programmers who work full time "cutting code" (Rettig, 1990).

JAD and other open paradigm models assume that the most important work of the team takes place in face-to-face interaction. Teams in the real world, especially on medium- to large-scale systems, are unlikely to do all their work face to face, although for small project teams this may be practical and even desirable. More typically, development teams mix independent and face-to-face activity. If we look at the pattern of team activity, traditional project teams might follow the pattern of Figure 1(a), never actually working together in one place at one time, except for kickoff and kudos, meeting once at the very beginning and possibly again at the end to celebrate or commiserate. The JAD team follows a pattern more like that of Figure 1(b), beginning in dispersed efforts, joining together for the JAD session itself, and then dispersing again. The activity of a structured open team is more likely to resemble Figure 1(c).

Closely coordinated teams may use various kinds of meetings to coordinate their work and improve its quality. These include racking, walkthroughs, problem resolution, and system reconciliation.

Tracking covers those regular meetings that keep all members of the team informed of progress and of events that affect the project. Tracking meetings may be the least productive (often the same purposes can be fulfilled through memos or email), but the face-to-face contact can be essential to maintaining a sense of "team-

Large companies supporting OOT

Steve Jobs' brilliant NextStep environment continues to rate very high in the software community, and it edged out some impressive competition to win this year's Andrew Fluegelman award. ... In the future, all desk tops will include a unified imaging model (which gives true what-you-see-is-what-you-get integration between display and print-outs). They'll also feature an object-oriented development system, a well-designed user interface with enhancements like 3-D buttons, aids for end user programming, support for voice annotation, and other advances. "NextStep is the first complete commercial system with all of this," notes developer Dan Bricklin, a Fluegelman judge. "It's a milestone."

Source: "NextStep Wins Andrew Fluegelman Award," PC WORLD, August 1990.

Anti-hype

Object-oriented programming and databases, which lie on the cutting edge of software technology, offer some long-term advantages over traditional methods. Object-oriented approaches are particularly well suited to some niche applications. ... But because the technology is still in its infancy, there are many pitfalls, and the economic cost-justification for object-oriented programming remains meager. Thus, wise MIS managers will apply some object-oriented techniques to particular applications, but forgo dependence on those techniques until the technology has had a chance to mature.

Source: "Object-Oriented Technology Fills Leading-Edge Niche," DIGITAL REVIEW, Gene Grygo, July 30, 1990.

Early users of object-oriented technology have found that the move to object-oriented programming does not force them to discard their older systems. Object-oriented databases and design are best known for providing necessary support for users with complex data modeling needs. But they can be used in conjunction with other data management technology to solve a wide variety of problems ... Dave Pendleton, chief of the Artificial Intelligence Group at National Charting within the National Oceanic and Atmospheric Administration (NOAA) in Rockville, Md., is using the technology to convert the creation and updating of nautical charts from a manual to an automated system. ...Before implementing Gemstone, an object-oriented database from Servio of Alameda, Calif., Pendleton had run into major obstacles. "Conventional techniques failed to provide the economic payoff," he said, because most of the problems had no algorithmic solution, no step-by-step process that procedure-oriented systems could adequately describe. ... "Just because we're using object-oriented for the compilation subsystem doesn't mean that we've thrown out relational," Pendleton said. ... "Object-oriented systems will ... live alongside conventional relational systems doing what they do best and letting other systems do what they do. It won't be displacement, but augmentation ... Object-oriented systems can handle more complicated problems."

Source: "Objects Augment New Development," DIGITAL REVIEW, Melinda-Carol Ballou, July 30, 1990.

Despite its obvious advantages for software development, object-oriented programming (OOP) has taken almost 15 years to catch on among developers, and still meets resistance in many quarters. One problem that slowed its adoption, especially in the DOS world, was the demands OOP languages placed on hardware. The introduction of chips like the 80386 and 80486 has largely eliminated this difficulty.

Source: "OOP Needs Changed Programmer Thinking," COMPUTER TECHNOLOGY REVIEW, Dave Trowbridge, August 1990.

Standards

According to some industry analysts, the benefits [of object-oriented databases] are so compelling that OOP systems will overtake procedural systems by the year 2000. ... Unfortunately, in today's world, each of the dozens of OOP vendors, most of them small start-ups, has designed a unique environment. This includes competing object-oriented database vendors such as Servio Corp., Versant Corp., Ontologic Inc., and Objectivity Inc. ... Someone, somewhere has got to come up with a set of standard interfaces to link object-oriented technology with the everyday procedural code most people use. ...

One group that thinks it can bridge the gap is the 1-year-old Object Management Group (OMG). ... "OMG is a technical and promotional vehicle," Slitz [John, OMG Marketing Vice-President] said. "The idea is to get the industry to accept a new paradigm, which is object-oriented technology. But since nothing in technology ever goes away, this paradigm is going to have to overlap that of procedural code."

Source: "They're Still Saying OOPs," COMPUTERWORLD, August 20, 1990.

News

Digital Equipment Corp. of Maynard, Mass., and Objectivity Inc., of Menlo Park, Calif., have entered into a joint development and marketing partnership for Objectivity/DB, an object-oriented database management system (OODBMS). As part of the deal, Objectivity/DB will be sub-licensed and distributed to end users by Digital. Both companies plan to work together within industry standard organizations to advance evolving standards for object-oriented systems. The announcement comes at the heels of DEC's recent corporate membership in the Object Management Group.

Source: SOFTWARE INDUSTRY REPORT , August 20, 1990.

Languages & tools

Object Professional from Turbo Power Software is a splendid library of object-oriented classes for Turbo Pascal 5.5 programmers. It's extensive — consisting of over 50 units, 130 object types, 2100 documented procedures, functions and methods, 1200 additional internal routines, and 1600 pages of text and examples neatly divided into 3 manuals. Object Professional includes complete Turbo Pascal and assembly language source code so you can modify, append and derive to you heart's delight. If you're a Turbo Pascal 5.5 programmer developing applications that need sophisticated screen management, data input/output, or TSR routines, check out this package. It could save you months of development time.

Source: "Inside Object Professional," DR. DOBB'S JOURNAL, Gary Entsminger, September 1990.

Solbourne Computer Inc. of Longmont, CO, has developed the OI (Object Interfaces) Tool Kit, an alternative tool kit licensed by AT&T USO for sale, distribution and support. According to Andrew Gerber, senior engineer at Solbourne, the product lets developers select either an OpenLook or Motif look and feel at run time. "We wrote the tool kit so we didn't have to be in the position of selecting Motif or OpenLook," he said. ... Developed in the object-oriented C++ language, the OI Tool Kit provides a generic set of objects, such as buttons, scroll bars and sliders, for UNIX GUI development, he said. "We've taken the Motif and OpenLook interfaces and mapped connections between them. You can write the application with the tool kit and select Motif or OpenLook at run time with the same executable [code]."

Source: "Supporting Two GUIs? New Tools Ease the Headaches," DIGITAL NEWS, Len Grzanka, August 20, 1990.

If the tide is beginning to turn in favor of those who advocate the Ada programming language for Defense Department systems, it is partly because many who originally resisted Ada are now among its most vocal proponents. ... This message was delivered recently at a conference here by Marilyn Andrulis, the chief executive officer of Andrulis Research Corp. in Bethesda, MD. ... Andrulis said that by 1995, half of all the code under development worldwide will be written in Ada. ... A variety of factors will increase Ada use she said, including DOD's affirmation of the Ada standard in its recently approved software master plan and the adoption of European standards by commercial interests in the United States, Japan and Canada. Ada, Andrulis noted, is the primary language for banking and finance systems in Europe. ... Andrulis said there has been progress in lowering two key barriers: the shortage of commercially obtainable Ada development tools and resistance from senior software engineers raised on Fortran, Cobol and earlier languages. ... But possibly most significant, she said, has been the "culture shock" experienced by senior software engineers forced to use Ada. Though some engineers will never adjust, Andrulis said those who do "become the most vocal and aggressive proponents of Ada and object-oriented development.

Source: "Wave of Proponents Turns the Tide in Favor of Ada," GOVERNMENT COMPUTER NEWS, Robert Green, August 20, 1990.

Turbo Pascal, C++ and Smalltalk do not directly support the saving of objects from one invocation of a program to another. This is like building a car every morning, driving it to work, and then having it disintegrate when you get back home at night. Every time you commute to work, you need to build a new car. Similarly, whenever you create an object in a program, it exists only until the program's work is done. Every time the program starts, it has to begin by building new objects. ... Bertrand Meyer incorporated the concept of "persistent objects" into his Eiffel programming language. A persistent object can be stored external to a program, usually in a file. More than one program can share the same persistent object, and a program can save its objects for future executions. The article goes on to compare, in technical detail, object implementation in Turbo Pascal vs. Smalltalk and C++.

Source: "Persistent Objects in Turbo Pascal," DR. DOBB'S JOURNAL, Scott Robert Ladd, September 1990.

Conference reports

The recent three-day SCOOP-Europe — Seminars and Conference in Object-Orientated Programming — session at Imperial College, London, drew enthusiastic delegates from all over Europe, many from commercial organizations that are beginning to realize there really could be concrete benefits to be gained from using object-oriented technology. Most were looking for answers to questions like why? Why use object-orientated techniques, what benefits can they bring? And how do you go about doing it? Although the seminars weighed on the technical and programming side, the best attended sessions heard speakers argue that traditional software development techniques can be profitably replaced by object-orientated alternatives. ... Jeff McKenna of the McKenna Consulting Group reviewed the use of object-orientated techniques (the object-orientated community call them OOTs) from a management point of view. "The main thing to remember," he says, "is the benefits from using OOTs accrue over time." When the object of the exercise is to produce reusable software, it will take two or three projects using OOTs to build up the experience and a library of sufficient size to draw upon for other projects, by which time McKenna claims that 80%–90% of a project's code should be reused from others, and little more that 10% newly written.

Source: "SCOOP-Europe Woos Commercial Users to Object-Orientated Technology" UNIGRAM X, August 10, 1990.

ness," especially in larger projects where people otherwise might not have occasion to be in the same room.

Walkthroughs are formal reviews of a problem statement, design, or piece of code in a meeting led by a facilitator or moderator and supported by a recorder. Although they can be used in many ways, walkthroughs have a special role in teams, where they permit large or dispersed teams to function more as if they were doing their work conjointly. In functioning teams, walkthroughs are neither formalities nor simply occasions for critiquing the work of others, but are the mechanism that enables everyone to build a sense of ownership and feel confident in the final product. Walkthroughs keep work out in the open, ensuring reliability and the highest standards of quality.

Problem resolution meetings are ad hoc assemblies of the entire team or an interested and involved subgroup of stakeholders in a particular problem. Such a meeting may be called whenever there is a problem that spans the work of multiple team members.

System reconciliation refers to the process of taking several independently derived designs or portions of design and reconciling their structures into a common object structure that maximizes object class reuse. Without such meetings, larger groups tend to miss opportunities for expanded reusability and for evolving a robust development framework.

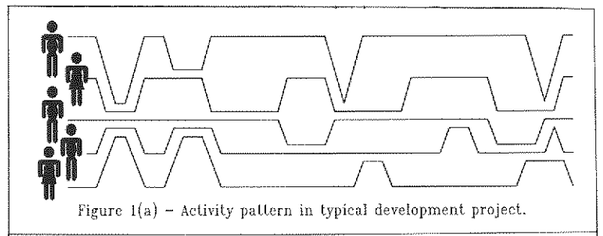


Figure 1(a) – Activity pattern in typical development project.

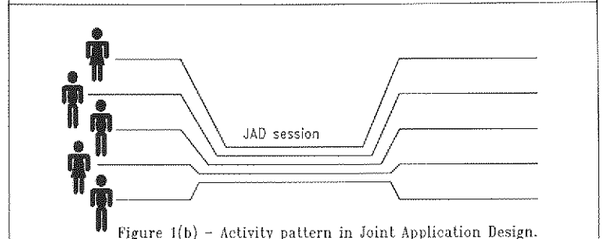


Figure 1(b) – Activity pattern in Joint Application Design.

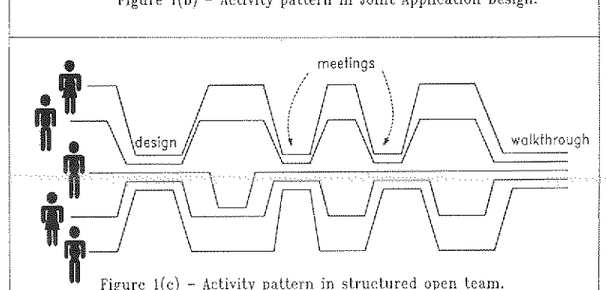


Figure 1(c) – Activity pattern in structured open team.

OBJECTIVE REQUIREMENTS

Object-oriented software development, despite the claims of a few extremists, has much in common with traditional approaches. It also raises some special emphases. Many of the special issues regarding teams arise from the complicated issue of building and using reusable artifacts (Goldberg and Rubin, 1990b).

Most experienced developers now see that developing framework software and reusable class libraries involves special skills and requires extra care and attention. Some have even proposed forming separate applications and framework teams (Goldberg and Rubin, 1990a). Others (Thomsett, for example) see this as best solved by giving project teams access to internal consultants specializing in the reuse repository. Such members may be assigned to a single team or may rotate among teams.

Thomsett's "linking pin" idea has also been proposed as an ongoing mechanism for assuring reuse across different subprojects in a large application. A linking pin is someone who splits their time between two or more teams and is organizationally responsible to each such team. In object-oriented development, one of the key functions of a linking pin member is to identify opportunities for expanded reusable class hierarchies and to maintain consistency in framework utilization.

Whether framework development precedes, follows, or is concurrent with application development, there are limitations and difficulties as well as gains. All object-oriented development teams must figure out what approach fits best with available resources, team organization, and project goals. A separate framework team may make the best use of specialized expertise, but introduces another interface to manage and does not facilitate skill dispersion. On the other hand, integrating framework and application developers into a single application team can lead to more chaotic development of the framework layer.

USERS AND REUSE

In many circles, users, clients, and applications experts are viewed as inconvenient facts of life, to be dealt with as little as necessary, preferably only until they've signed off on a requirements document, after which their phone calls are seldom returned. In object-oriented development, however, close modeling of an application must be carried from requirements definition and problem analysis all the way through implementation. A good piece of object-oriented code is also a good model of a portion of the applications domain. Close, robust modeling of the underlying conceptual structure enhances reusability, extensibility, and maintainability, which means that even during coding, late decisions may require application domain knowledge to be made well.

In prototyping or other modified development cycles with less-than-thorough designs, this need is even more acute. Increasingly, experience points to the need for continuous access to application expertise throughout the object-oriented development cycle. This isn't as much a problem for small systems produced by amateurs or user-programmers or for programming tools produced by programmers; in these cases, the programmer is an application domain expert. However, for larger systems where programmers work on systems in other applications areas, it can be a major problem.

Teams create a framework for bringing applications domain experts into the entire development process as team members or consultants. Joint Application Design teams accomplish this through joint user-developer teams for analysis and system design, but then leave program and detailed design as well as implementation to the developers. In the structured open team, a user or applications expert is made an integral part of the development team "for the duration" whenever possible or remains "on-call" otherwise.

ALL TOGETHER NOW

The pioneering work on Joint Application Design and structured open teams is only the beginning. In many respects, teamwork models for software development are in their infancy. Most established "methodologies" arose in the era of small systems or perpetuate the assumptions and biases of that era. They were not devised with teams in mind nor with explicit recognition that most software development is a group activity even though the solitary programmer remains for many the mythic ideal. The next decade, through expanded use of such technology as groupware, centralized object repositories, and networked CASE tools, through better understanding of group and organizational dynamics, and through practical validation and elaboration of new teamwork models, is likely to see the emergence of powerful team-oriented methodologies that thoroughly integrate the elegance of object-oriented modeling with the synergy of effective teamwork. ≡≡

References

Constantine, L.L. *Family Paradigms*. New York: Guilford Press, 1986.

Constantine, L. L. "Teamwork Paradigms and the Structured Open Team." *Proceedings: Embedded Systems Conference*. San Francisco: Miller Freeman, 1989

Constantine, L. L. "Organization Paradigms and the Management of Change." From *Proceedings: Software Development '90*. San Francisco: Miller Freeman, 1990.

Doyle, M., and Strauss, M. *How to Make Meetings Work*. New York: Jove, 1982.

Goldberg, A., and Rubin, K. "Talking to Project Managers: Case Studies in Prototyping." *Hotline on Object-Oriented Technology*, 1(8): 5–8, 1990a.

Goldberg, A., and Rubin, K. "Talking to Project Managers: Introducing Reuse into the Development Process." *Hotline on Object-Oriented Technology*, 1(9): 11–14, 1990b.

Rettig, M. "The Practical Programmer: Software Teams." *Communications of the ACM*, 33(10), October 1990.

Straus, J., and Spiwack, D. "The IS Productivity Challenge: Motivating Your People to Produce Breakthroughs." *American Programmer*, 3(7/8): 57–63, 1990.

Thomsett, R. "Effective Project Teams: A Dilemma, A Model, A Solution." *American Programmer*, 3(7/8): 25–35, 1990.

Thomsett, R. *People and Project Management*. Englewood Cliffs, N.J.: Prentice-Hall, 1980.

Wirfs-Brock, R., Wilkerson, B., and Wiener, L. *Designing Object-Oriented Software*. Englewood Cliffs, N.J.: Prentice-Hall, 1990.

Wood, J., & Silver, D. *Joint Application Design*. New York: Wiley, 1989.

Zahniser, R. A. "Building Software in Groups." *American Programmer*, 3(7/8): 50–56, 1990.

Larry L. Constantine is a widely published software methodologist and an independent organizations and systems consultant. He researches and teaches object-oriented design and teamwork organization and chairs the Object-Oriented Systems Symposium, as well as People and Systems: The Conference on Organization and Management in Systems Development. Constantine can be reached through Internet at 72067.2631@compuserve.com and by phone at (508)264-4274.

FYI

Excerpts from leading industry publications on aspects of object-oriented technology

News

Why is there so much fuss about object-oriented programming? As Dave Thomas, president of Object Technology International Inc. of Ottawa, puts it, "We're screaming for help. Software development has gotten so enormously complex that we desperately need some sort of sensible software engineering process." ... Thomas recently addressed the Emerging Technology series of Teleforum hosted by NDMA Inc., of Ridgefield, CT. The session was attended by 70 information executives in the US and Canada. ... Thomas also expects an object COBOL — probably still two or three years out for serious use; object Ada in 1995–97; and environments for using C++ that make it almost as good as Smalltalk, in two or three years. He also expects real object databases, as well as widespread industrial use of Smalltalk.

Source: "'We're Screaming for Help' Says Dave Thomas Who Heralds Object-Oriented Programming," SOFTWARE INDUSTRY REPORT, August 20, 1990.

Cox [Brad, co-founder and chief technical officer of Stepstone Corp.] argues that software development, currently a technical discourse between a solitary programmer and a computer in which the design process, the producer and the programming tools are supreme, will be transformed into a commercial matter between software producers and the consumer, in which the product, the consumer and components are emphasized. In short, a social activity where the customer's wishes are fully—and not unreasonably—represented.

Source: "In the Software Revolution, object orientation 'Will Make Common Sense Apply'," UNIGRAM X, William Fellows, August 17, 1990.

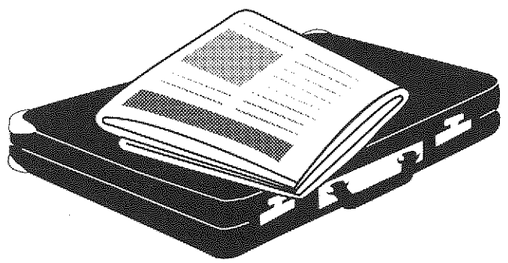
Applications

The United States Geological Survey has boosted its power for processing topographic maps by purchasing 40 Unix-based Intergraph workstations. ... The Intergraph systems, called Standalone Edit Stations (SES), will be used at 5 USGS centers to collect, process and disseminate topographic map data in both graphic and digital form. The systems have been linked to the USGS' Gould superminicomputer through an existing local-area network. These systems run Intergraph's specialized object-oriented software, TIGRIS. ... By using the TIGRIS system, USGS geologists can structure the data so that geometry, topology and other attributes are integrated into a single file, said Grady [Rich, executive manager in Intergraph's federal system division]. Since the user is not going between two file structures, he or she has greater control over the data integrity.

Source: "Map-Making via UNIX," UNIX TODAY, Lee Bruno, August 6, 1990.

The Macintosh lets users manipulate graphic screen elements with a mouse or keyboard. A typical Macintosh application will feature one or more windows or related text and graphic screen elements, such as those used to edit text. Since the burden of managing this complexity is left to the programmer, even simple Macintosh applications are prone to bugs and are difficult to maintain. Object-oriented programming techniques can help reduce this complexity. In this article, I use standard C to define window objects as part of a generic application. Any Macintosh program derived from these templates will easily support many windows of several different types.

Source: "Using OOPs in Mac Designs," THE C USERS' JOURNAL, Richard Rathe, August 1990.



BOOK REVIEW

DESIGNING OBJECT-ORIENTED SOFTWARE

by Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener
Prentice-Hall, Englewood Cliffs, NJ, 1990; 341 pages

Reviewed by Douglas W. Bennett

The authors make a significant contribution to the state of the art of designing object-oriented programs. While they make, then back away from, some very significant advances to the state of the art, it is still more practical than many other books in the field.The purposes of the book are, to quote from the preface, to "... offer basic design principles and a specific design process that can be applied to any software programming effort ... provide a coherent model for the design process: responsibility-driven design ... provide tools, such as the [hierarchy and collaborations graphs], to help the designer every step of the way."

The book provides a fairly complete and informal process (paper and pencil notation) for identifying the objects that make up a system and allocating behavior to them. Such an approach is an appropriate next step for organizations developing O-O systems with no design documentation (probably the great majority).

The opening chapters give a very good introduction to the terms and ideas used in the O-O community. Chapters 3 through 8 lead us through the process of developing complete specifications for all classes needed for an application, using an automatic teller machine design as the example. A second complete example (an online documentation system) is worked through in chapter 10. The main steps are to identify classes; allocate responsibilities or behavior to them; identify collaborations between classes needed to meet the user requirements; organize classes into inheritance hierarchies and identify contracts, or groups of responsibilities; group classes into subsystems; and, finally, formalize responsibilities by identifying the protocol, or messages, needed to carry out the contracts between classes. A chapter on implementing the design concludes the description of the process. A quick reference to the process and notations and the complete documentation for both examples are provided in appendices.

The strong points of the book are the examples; the detail used to describe the process at each stage; language independence; provision of several useful notations; and the emphasis on characterizing object behavior in terms of interactions between objects as collaborations and contracts. A welcome addition to the process is the suggestion that responsibilities of a class can be reallocated to improve the modularity of the design.

There are several problems in the book. One is an overemphasis on inheritance as the mechanism for building objects at the expense of composition–construction–ownership (composition is not mentioned until chapter 9). In O-O languages, composition and inheritance are two ways of implementing a useful object. Bertrand Meyer [OBJECT-ORIENTED SOFTWARE CONSTRUCTION, Prentice Hall, 1988] has elegantly stated the limits of inheritance as an implementation tool. Another problem is in the handling of client and server behavior. The idea of having client and server ends of a contract, or collaboration, between two objects is a good idea that is handled well. The effectiveness of the distinction is reduced, however, when the authors suggest that each object can be a client in some collaborations and a server in others. The modularity of the objects suffers as a result.

The big advance that the authors almost make and then back away from is the idea of subsystems. Subsystems are defined on p. 135 as "... groups of classes ... that collaborate among themselves to support a set of contracts. From outside the subsystem, the group of classes can be viewed as working closely together to provide a clearly delimited unit of functionality." In other words, subsystems are a higher level of object. This would be a significant advance over the common idea that one level of component — an object in an OOPL — is sufficient for building large, complex systems. They back away from taking full advantage of the advance when they say on p. 136, "But subsystems are only conceptual entities; they do not exist during execution. They cannot directly fulfill any of their contracts. Instead, [they] delegate each contract to a class within them that actually supports the contract." That is like saying that the memory-CPU board in a computer is only a conceptual entity because the only things that really do any work are the chips on the board. In my opinion, multiple levels of objects — classes and subsystems — are needed if object technology is going to make a serious contribution to handling complexity in large systems.

In spite of my complaints, I feel this book is the most useful and practical description of a design process for O-O programs I have seen. To indicate that this is not an idle opinion, I will begin using this book as a supplementary text in my course on O-O software engineering. ≡≡

Douglas W. Bennett works at DesignWays, which offers training and consulting services aimed at improving processes and organizations that develop software products. He can be reached at DesignWays, P.O. Box 581, Newtown, CT 06470, (203)426-9511.

DATABASES

Race for the database pennant: a third-generation manifesto

Dr. Jacob Stein

In response to The Object-Oriented Database System Manifesto, the Committee for Advanced DBMS Function has published the Third-Generation Database System Manifesto.

Members of the committee include Michael Stonebraker of UC Berkeley; Lawrence A. Rowe of UC Berkeley; Bruce Lindsay of IBM Research; James Gray of Tandem Computers; Michael Carey of the University of Wisconsin; Michael Brodie of GTE Laboratories; Philip Bernstein of DEC; and David Beech of Oracle Corporation.

The committee's manifesto provides both a relational perspective on the Object-Oriented Manifesto and the opinion of respected relational practitioners on the required characteristics of next-generation DBMSs.

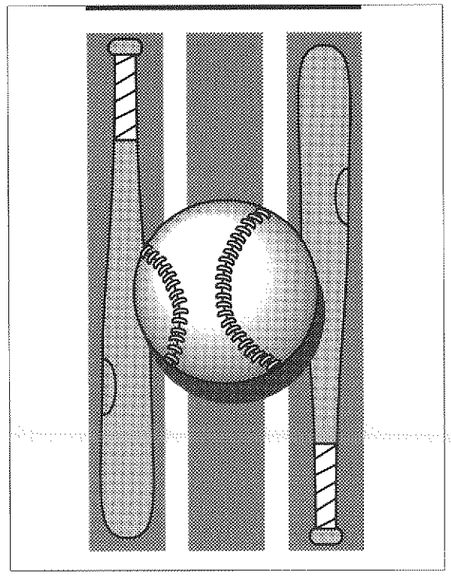
This article will serve as a guide to comparing and contrasting the two manifestos. In a future issue of THE HOTLINE, we will examine the Third-Generation Manifesto in greater detail. The Object-Oriented Manifesto is organized by thirteen Golden Rules and five optional features:

Rule 1: Thou shalt support complex objects.
Rule 2: Thou shalt support object identity.
Rule 3: Thou shalt encapsulate thy objects.
Rule 4: Though shalt support types or classes.
Rule 5: Thy classes or types shall inherit from their ancestors.
Rule 6: Thou shalt not bind prematurely.
Rule 7: Thou shalt be computationally complete.
Rule 8: Thou shalt be extensible.
Rule 9: Thou shalt remember thy data.
Rule 10: Thou shalt manage very large databases.
Rule 11: Thou shalt accept concurrent users.
Rule 12: Thou shalt recover from hardware and software failures.
Rule 13: Thou shalt have a simple way of querying data.

Feature 1: Multiple Inheritance.
Feature 2: Type checking and type inferencing.
Feature 3: Distribution.
Feature 4: Design transactions.
Feature 5: Versions.

The first eight rules define object-oriented, and the last five rules define a DBMS. The optional features are not required of a system for it to be an object-oriented database.

From a reading of the Object-Oriented Manifesto (see sidebar on page 10), it is apparent that the authors were trying to define what an OODBMS is, not what additional features the market will require for commercial viability. At the time it was written, there was still a need to get programming language people to understand what makes an object-oriented system a database, and to get database people to understand what makes a database object-oriented. Hence, the object-oriented rules probably seem obvious to someone familiar with



Dr. Jacob Stein is a Corporate Staff Technologist at Servio Corporation. He has been involved in the research and development of object database management systems for nearly a decade and has lectured extensively and published numerous articles on object data management. Stein received his Ph.D. from the State University of New York at Stony Brook, where he worked on structurally object-oriented extensions to the relational model.

object-oriented programming; the database rules, obvious to someone familiar with databases.

The authors of the Third-Generation Manifesto appear to have a slightly different goal in mind. In addition to discussing the requirements for a third-generation data model, they emphasize database features independent of a particular data model, be it object-oriented, relational, or otherwise. For this reason, the two manifestos are not always directly comparable. The absence of a feature felt to be critical by The Committee for Advanced DBMS Function in the Object-Oriented Manifesto does not imply that its authors, or the OODBMS vendor community, thinks that they are unimportant. Indeed, I would dare to say that they also agree with the correctness and importance of many, though clearly not all, of the propositions put forth by the committee. However, they might be likely to disagree with much of the detailed discussion regarding the propositions.

> ***A comparison of the Object-Oriented vs. the Third-Generation Manifesto is surprising … they are far more often in agreement than disagreement, although some of the issues on which they disagree are significant.***

To the authors of the Third-Generation Manifesto, hierarchical and network DBMSs are first-generation systems and current relational systems are second-generation systems. The manifesto is organized by three tenets and thirteen propositions that the authors believe follow from the tenets. The Three Tenets are:

Tenet 1: Besides traditional data management services, third-generation DBMSs will provide support for richer object structures and rules.

Tenet 2: Third-generation DBMSs must subsume second-generation DBMSs.

Tenet 3: Third-generation DBMSs must be open to other systems.

Only the first tenet is directly addressed by the Object-Oriented Manifesto. The second tenet argues for the preservation of nonprocedural access and data independence, two of the most useful features of second generation DBMSs, in any system purporting to be third generation. The third tenet argues that a DBMS with broad applicability must have a fourth-generation language, provide access from multiple languages and to outside systems (e.g., spreadsheets), interface to graphics packages, and be distributed.

The third tenet is a good example of how misleading it can be to directly compare the two manifestos can be misleading. While the Object-Oriented Manifesto does not address the issues this tenet raises, every OODBMS vendor recognizes its importance and is actively addressing the features necessary for broad applicability of their products. To a large degree, the third tenet is a "mom and apple pie" tenet: it does not distinguish a third-generation database from an object-oriented database and database vendors of every ilk recognize its importance.

THE PROPOSITIONS

A comparison of the Object-Oriented Manifesto's rules with the Third-Generation Manifesto's propositions is surprising in that they are far more often in agreement than disagreement, although some of the issues on which they disagree are significant.

Proposition 1.1: *A third-generation DBMS must have a rich type system.* This proposition corresponds to Rule 1. In my opinion, both manifestos are in agreement here.

Proposition 1.2: *Inheritance is a good idea.* This proposition corresponds to Rule 5. The committee feels strongly that multiple inheritance is essential; the Object-Oriented Manifesto lists multiple inheritance as an optional feature. The issue of whether multiple inheritance solves more problems than it introduces is a religious one that I will not go into here. Suffice it to say that this hardly qualifies as a major distinction between the two manifestos.

Proposition 1.3: *Functions, including database procedures and methods, and encapsulation are a good idea.* This proposition corresponds to Rule 3 and relates to Rule 7. The committee feels that functions should be written in a higher-level language. If a computationally complete higher-level language is used, this addresses the requirement of Rule 7 for computational completeness. While it is possible that some 4GLs are computationally complete, it is stretching things to conclude that they are appropriate implementation vehicles for anything other than forms and menu-based applications.

The committee also feels that functions should access the DBMS through a high-level nonprocedural access language. When possible and bearable, this is desirable. But try writing a graph traversal algorithm using embedded SQL in C and you may not find it bearable. Many of the applications in which OODBMSs are being used today are not amenable to the use of high-level nonprocedural languages within functions. As this situation changes (either because of a change in the applications using OODBMS or because SQL is extended),

DOWN UNDER≡≡

The Australian object-oriented scene

Dr. B. Henderson-Sellers

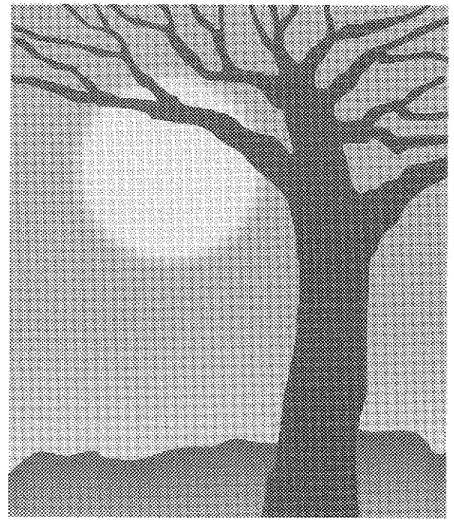
Membership of the Object-Oriented Special Interest Group (SIG) of the Australian Computer Society is growing rapidly. Both groups are now establishing themselves within the industrial and commercial environment in Melbourne and Sydney.

Discussions at a recent meeting of the SIG in Sydney ranged extensively across a wide spectrum of object-oriented management issues giving everyone present (including the speaker) "food for thought" regarding this important, yet often neglected, aspect of the object-oriented paradigm.

Interest in SIGs such as these and in the training courses being run by several organizations, sometimes linked with a product launch, is now coming from a wide spread of commercial organizations. Attendees span the whole spectrum from novices who have been made aware of the potential importance of object-oriented technologies in their careers to those who have been practising these concepts for some years but wish either to update or formalize their knowledge or to expand their expertise, say, from an object-oriented programming environment to acquire object-oriented analysis and design skills.

Perhaps the most significant event to be staged in Australia in the object-oriented arena will be TOOLS Pacific '90, which will be held in Sydney, in the new Darling Harbour complex, over the last three days in November. Paralleling the TOOLS conferences held in France, this symposium is aimed at bringing together developers of methodologies, ideas, and software tools and consumers of these techniques.

The Chairman of that conference committee, Dr. John Potter from the University of Technology, Sydney, is looking forward to a stimulating and enjoyable meeting that, it is anticipated, will raise even further the profile of object-oriented technology as a "reality" within the business concerns of Australia. I intend to include an appraisal of both that meeting and an Australian perspective on the OOPSLA meeting held in Ottawa in October in a later column in this journal. ≡ ≡



Dr. B. Henderson-Sellers is a senior lecturer at the School of Information Systems, University of New South Wales, Australia.

uisites). Same as the design prototyper, but typically with several more years work experience. A possible promotion chain is to move from design prototyper to reuse evaluator to designer.

Basis for evaluation. The reuse evaluator's goal is to identify and propose new components to be incorporated into the corporation's library. Thus, this person is evaluated on the completeness, accuracy, and ingenuity of the test suites used to test individual components. In addition, this person is evaluated on his or her decisions as to which components to test. Since there are many potentially reusable components and testing all of them may not be feasible, this person (in conjunction with the reuse librarian) is responsible for choosing those components whose reusability potential (in either a positive or negative sense) stands to have the greatest impact on the corporation.

Title: Reuse Librarian or Reuse Manager*

Hiring manager. Depends on the potential scope of usage of the library that is to be managed. Within an organization, there might be several levels of shared libraries, each with its own reuse manager responsible at the appropriate level. At the highest level (the corporation), this person is hired by the Chief Information Officer or Information Systems Manager. An individual project or a group of projects within a department might each manage libraries, and the hiring manager would be the project development manager or department manager.

Functional job description. Responsible for classifying, documenting, and entering new reusable components into the corporate or project repository. He or she:

- constantly reviews software for entrance into the repository, working especially with the reuse testers;
- acquires components from external commercial sources, and modifies them according to the organization's documentation and format guidelines;
- provides consulting to the analysis prototypers on the available reusable components that might be leveraged;
- can participate in the reviews for design prototypes to make sure that reuse is properly planned;
- can bring management ideas for generalizing applications into frameworks.

The contents of the repository have to be carefully controlled so that only components that have passed special review are included, and so that the external interfaces are consistently maintained once components are added to the repository. Documentation for each component should include indications of the context for which the component was designed, ways in which the component has already been reused, and a history of modifications.

Individual skills. A keen ability to take a new object and understand its relationship to a large body of existing objects. This involves excellent classification and abstraction skills. Must have technical competence in the technology used to construct the library of components. Should be a skilled communicator expected to participate in many reviews and work with each project's reuse testers to identify where reuse can be properly leveraged. In addition, must be an active industry watcher in order in order to procure reusable components from commercial sources when appropriate.

Qualifications (experience and educational prerequisites). Experience in organizing and utilizing large volumes of data with many potential connection complexities. Prior experience using the corporation's language and development tools. Excellent interpersonal skills as well as excellent verbal and written communication skills. Background in a field similar to that of components contained in the library to be managed. Educational background should be an MS in a natural science or computer science.

More often than not, the reuse manager has previously been a part of the company, moving through various programming levels or team-coordinating type tasks. Part of the skill set can also be found in head librarians or a highly skilled information officer. However, such individuals must have significant technical exposure as well.

Basis for evaluation. The reuse manager is primarily evaluated on how well the contents of the library support the ability of the analysis prototypers to create minimal software to rapidly produce prototypes. The quality of the library is measured by whether developers are willing and able to incorporate software found in the library into actual products, and how well tested they find this software to be (that is, how many defects are reported). The reuse manager is also evaluated on whether the library is growing appropriately to meet the development needs of the company. This is a subjective evaluation, tied to the variety of projects carried out; but it is also an objective evaluation based on lowering the costs of development over time when the variety stabilizes.

CONCLUSION

In this article, we have provided descriptions for the jobs associated with an object-oriented development effort and not normally associated with a traditional development effort. Although these descriptions accurately reflect the jobs as we view them, a given organization is likely to customize each description to better suit its needs. ≡≡

* In the September issue of THE HOTLINE, the Editor suggested that the title of Reuse Librarian is possibly not the ideal. Apparently, the connotation of the word "Librarian" is not sufficiently powerful or perhaps does not denote someone with the technical skills that are actually needed.

OODBMS support for high-level nonprocedural access will also improve.

Proposition 1.4: *Unique identifiers for records should be assigned by the DBMS only if a user-defined primary key is not available.* This proposition addresses Rule 5. My opinion is that update through a cursor does not constitute relational support of identity. User-defined primary keys fail miserably in situations such as representing physical connectivity in a parts hierarchy. OODBMSs rely too heavily on object identity. I predict that OODBMS vendors will be adding more value-based support and that the relational vendors will keep telling us that we don't really need identity.

Proposition 1.5: *Rules (triggers, constraints) will become a major feature in future systems. They should not be associated with a specific function or collection.* This proposition is not addressed in the Object-Oriented Manifesto. Yes, triggers and constraints will become a major feature in future systems. This is an area in which both relational and object-oriented vendors will need to do a considerable amount of work. I do not necessarily agree with the argument that rules should be independent of functions. In a system that enforces encapsulation, the maintenance nightmare envisioned by the committee will not arise.

Proposition 2.1: *Essentially, all programmatic access to a database should be through a nonprocedural, high-level access language.* This proposition corresponds to Rule 13. While the Object-Oriented Manifesto does not emphasize the point to anywhere near the same degree, bear in mind that its purpose was definitional, not feature-oriented.

With regard to the alternative to nonprocedural, high-level access, OODBMSs solve the single greatest problem with navigational access exhibited by first-generation systems. Namely, in first-generation systems the pointers one navigated through were physical disk addresses. The use of physical disk addresses made database reorganization extremely difficult. Perhaps the single most important contribution of second-generation systems was that of physical data independence. By using value-based references, these systems allow for data reorganization at the price of performance. By using logical, not physical, pointers, OODBMSs offer the promise of both physical data independence and high performance in many applications. Yes, some of this performance comes from taking advantage of recent changes in machine architectures and operating systems. When will the relational vendors do the same?

I do not necessarily buy the committee's claim that logical navigation will cause schema modification to be harder in OODBMSs. Schema modification is extremely difficult regardless of the DBMS in question.

Proposition 2.2: *There should be at least two ways to specify collections, one using enumeration and one using the query language to specify membership.* I would rephrase part of the proposition. In addition to enumeration, there should be a predicate-based means of specifying collections. Using a DBMS query language is one way of doing so. This proposition is addressed by the Object-Oriented Manifesto in its discussion of Rule 5. As the query languages of OODBMSs mature, they will incorporate this feature.

Proposition 2.3: *Updatable views are essential.* We are all in a lot of trouble here. While second-generation systems provide limited support for updatable views, a fully general solution to the view update problem will be extremely difficult if not impossible to find, regardless of the data model in question. OODBMS vendors are already addressing one of the important problems views address, access control.

> ***For better or worse, SQL is intergalactic dataspeak ... yes, and object-oriented SQL will become a reality.***

Proposition 2.4: *Performance indicators have almost nothing to do with data models and must not appear in them.* I have no disagreement with this proposition. However, OODBMSs are being used in application areas not well addressed by second-generation systems. OODBMSs provide a data model well suited to these applications and have optimized their systems to the needs of these applications. Second-generation systems have a data model and optimizations tailored for a different set of requirements, e.g., on-line transaction processing.

Proposition 3.1: *Third-generation DBMSs must be accessible from multiple higher-level languages.* Yes, but the fact that some OODBMS vendors have chosen to focus initially on a single language is more a reflection of their initial target market than a belief in a single language for all applications. As the market matures, OODBMS vendors will provide access from multiple higher-level languages. Already, GemStone from Servio provides language bindings for C++ and Smalltalk, allowing access to the same database from both languages.

Proposition 3.2: *Persistent X for a variety of X's is a good idea. They will all be supported on top of a single DBMS by compiler extensions and a (more or less) complex runtime system.* I think the committee's point here is that providing multiple language bindings that work well to-

gether will be difficult. True, but it will be nowhere near as difficult as finding a fully general solution to the view update problem. The problem is solvable and there is already progress in this area (see Proposition 3.1 above).

Proposition 3.3: *For better or worse, SQL is intergalactic dataspeak.* Yes, and object-oriented SQL will become a reality. There is already movement in this direction, which will increase as OODBMSs expand from engineering to commercial marketplaces. Many of the features of object-oriented SQL will also make for a better relational SQL. It may also serve as the key to integrating second-generation and object-oriented DBMSs.

Bear in mind that while SQL may be intergalactic dataspeak, the number of dialects out there greatly dilute its usefulness as a standard. As Stu Madnick of MIT points out, generally only 30% of the SQL used in applications is standard.

Proposition 3.4: *Queries and their resulting answers should be the lowest level of communication between a client and a server.* I don't understand the committee's point here. OODBMS proponents are not debating whether query requests "should be for single records, single pages, or some other mechanism." We are debating these issues with regards to implementation concerns, but we are not suggesting that users be forced to deal with these issues at the query level.

CONCLUSION

While the Third-Generation Manifesto raises several important issues, I must question the method of presentation. The authors seem to often confuse issues relating to data models with those relating to implementation and features independent of the data model. It is this confusion that we will attempt to resolve in our next article.

To me, and I believe most of the OODBMS community, the question isn't *which is best?* but rather *when is each most appropriate?* In my vision of the future, many organizations will be running both second-generation and object-oriented DBMSs, perhaps tied together through an object-oriented SQL.

The Object-Oriented Database System Manifesto was published in THE PROCEEDINGS OF THE DEDUCTIVE OBJECT-ORIENTED DATABASE CONFERENCE in Kyoto in December, 1989. The Third-Generation Database System Manifesto is available from the Electronic Research Laboratory in the College of Engineering at the University of California, Berkeley, as Memorandum No. UCB/ERL M90/28. If you would like a copy of either or both of the manifestos, please write to me. ≡≡

The Object-Oriented Database System Manifesto was published by a distinguished group of OODBMS researchers, including:

Malcolm Atkinson of the University of Glasgow;
Francois Banchilon of Gip-Altair;
David DeWitt of the University of Wisconsin;
Klaus Dittrich of the University of Zurich;
David Maier of Oregon Graduate Center; and
Stanley Zdonik of Brown University.

OBJECT-ORIENTED GOLDEN RULES

Rule 1: Thou shalt support complex objects

It must be possible to build complex objects from other objects and primitive types via "object constructors" or "type generators." It must be possible to represent tuples (or records), sets, bags, lists, and arrays as object constructors. It must be possible to store any object in any of these type generators. Language constructs such as C structs and Pascal records satisfy this requirement (the relational model does not satisfy this requirement, as the only sets allowed are sets of tuples). It must be possible to define operators that traverse complex objects.

Rule 2: Thou shalt support object identity

Complex objects shall have an identity independent of the values they contain, which allows objects to be shared. Typically, programming languages embody the notion of object identity in pointers. Object identity gives rise to two levels of object equality: objects may be identical, or they may contain the same fields. The first level is computed by comparing pointers to the two objects, while the second level requires comparison of the fields of the two objects.

Rule 3: Thou shalt encapsulate thy objects

One of the familiar principles of object-oriented technology is the separation of the object interface and the object implementation. The object interface consists of a set of functions providing a formal definition of the contract to be satisfied by the object. The object implementation consists of functions and states that are completely invisible to the clients of the object. This notion of encapsulation allows the object implementation to change in radical ways, without changing the way in which any application uses the object. Languages like C++, which provide for a distinction between public and private members, provide encapsulation.

Rule 4: Thou shalt support types or classes

As long as encapsulation is supported, a language may either support declaration of data types that statically describe the make-up of an object, or support the more dynamic notion of a class, in which the object creation paradigm is to make copies of existing objects. One should not be confused by the word "class": C++ classes are types in the terminology of the paper.

Rule 5: Thy classes or types shall inherit from their ancestors

Substitution inheritance requires that any operation on an ancestor is supported on a descendant. Inclusion inheritance requires that any descendant type be a subtype of all of its ancestors. Constraint inheritance provides for definition of subsets of a type, such that every element of the subset satisfies some constraint. Specialization inheritance occurs when a subtype is created by "specializing," or adding information to, elements of the ancestor type. Any of these models can satisfy the rule, and essentially all object-oriented systems provide one or more of these mechanisms.

thinker, one who understands the dynamics and mechanics of an architectural framework. Furthermore, although this prototyper understands the details of implementation requirements, he or she can overcome the desire to polish the prototype code, and instead can focus on how to organize requirements into conceptual categories. By generating such categories, the prototyper may propose new reusable components for the organization's library.

Qualifications (experience and educational prerequisites). Excellent programmer in the company's selected implementation language and environment, or ability to learn quickly. Educational background requirements vary with the company's overall expectations for its programming staff. Not a 4GL programmer, but rather someone with systems or applications development experience.

For example, a company might require that the person already be familiar with C++ and related Unix or PC/Mac tools, or with Smalltalk. That is, the person should be adept at the company's selected object-oriented language and environment.

At ParcPlace, we would look for a BS, preferably in Computing Science, with two years' work experience as a product development programmer, or an MS in Computing Science, or at least four to five years experience as a product development programmer. The kind of programming background depends on the company's own product plan, for example, whether someone with graphics, user interface, or database development is desired.

Basis for evaluation. The designer and design prototyper are evaluated on the ease with which the design is translated to implementation, the flexibility with which the design can be modified, and the extent to which the design allows the implementor to reuse components from the corporation's repository. In addition, design prototypers are evaluated on their contributions to the repository, whereas coders are evaluated on their withdrawals from that library.

Title: Reuse Evaluator

Hiring manager. Product Development Manager or technical person responsible for overall management of a particular software project, where this project is contracted or initiated internally.

In some cases, reuse evaluators might report to the company's reuse librarian. Note that in some companies, evaluators or testers report to a quality assurance manager, who reports outside the engineering development department. If so, this could be the same for the reuse evaluator, although in this special case we recommend not doing so. The reason is that the reuse evaluator has to have a special relationship with the designer and design prototyper to help this team meet the additional requirement that reusable components be created.

Functional job description. Determine whether or not software components have been designed for broad applicability.

Whereas the design prototyper is evaluated on an ability to create reusable components, it is the reuse evaluator who provides the special testing of these components by verifying their potential wide-ranging applicability. Thus, it is the combination of reuse evaluator and design prototyper that recommends new entries into the repository of reusable components. Reuse evaluation does not eliminate the need for traditional testing, the purpose of which is to verify whether the components work according to specification.

> ***A possible promotion chain is to move from design prototyper to reuse evaluator to designer.***

When the team goal is to create an application framework, the team assignments are modified to take into account the creation and testing of both the application framework and library of reusable components.

- The Framework Tester is responsible for testing the reusability of the designer's framework. This requires the tester to develop typical and atypical usage scenarios to evaluate the reuse effectiveness of the framework.
- The Components Tester is responsible for testing whether the components interact correctly with the existing framework and whether the components are independently reusable for other domains.

Individual skills. Excellent programmer, with the special skill of being able to think about unusual ways to apply existing software. Able to scout around the corporation to learn about prior, current, and expected future software development needs. A highly creative and inquisitive person, able to imagine projects that do not exist and be able to test the components for these possibilities. This means programming parts of such projects to see how the proposed reusable component might fit.

The reuse evaluator is one of the people with 20/20 hindsight and 20/20 foresight, because he or she will be asked to recall how a prior project could have benefited from a proposed new reusable component, and to predict how future projects of the corporation could benefit as well. It is important to note that the reuse evaluator is an apprentice, in the long term, for the job of designer.

Qualifications (experience and educational prereq-

ular software project, independent of whether the project is contracted or initiated internally.

Functional job description. Programs the executable prototypes during the analysis phase. The prototyper works with the analyzer to create an executable version of the proposed set of behaviors. The analyzer is responsible for deriving a complete and accurate statement of the system. This involves interviewing end users already familiar with the system as well as examining existing documents and documents or references that describe the discipline or disciplines that the system models. The analysis prototyper also works with end users to receive feedback, which may influence subsequent modifications.

The analysis prototyper is concerned solely with developing the demonstration of the proposed behavioral requirements, not with the quality of the implementation code. To that end, this person should attempt to reuse as many software components or frameworks as possible to get the task done quickly. Neither the analyzer nor the analysis prototyper should be concerned with the back-end technology.

Parenthetically, this job could be obsolete if the analyzer has a set of tools that allows him or her to enter the requirements specification into a system that automatically generates the executable form. This is typical today of data dictionary and restricted screen painting (4GL) facilities. The analysis prototyper and analyst could also be the same person if the person doing the analysis is a competent programmer who finds it easier to think through a consistent analysis by describing the behaviors in a programming language.

Individual skills. Need not be a sophisticated programmer if 4GL-like tools for the object-oriented prototyping system are available. (For example, a visual-interface construction kit would be useful in constructing a mockup of the desired application and in providing the end user with an executable proof-of-concept.) Must actually create the prototype, so should be able to place secondary relevance on details such as performance, architectural quality, and size, as these are not essential during this phase of development.

Qualifications (experience and educational prerequisites). Ability to program in the company's selected prototyping language and environment. Educational background requirements vary with the company's overall expectations for its programming staff. Should be personable, able to interact well with nontechnical people, especially if end-user interaction is expected.

For example, a company might require that the person already be familiar with Information Builder's Focus 4GL, or Easel, a screen painting system for PCs. As another example, this person might have to know Smalltalk, or be able to use a screen painting 4GL such as Facets, a 4GL application built on Smalltalk. At Parc-Place, we would look for a BS, preferably in Computing Science, or someone with at least four years' experience as a programmer in a corporate information systems group or database development group.

Market factors. Analysis prototypers can often be found in large corporate information systems departments, where quick turnaround applications are built using 4GLs. Often, someone fresh out of a computing science program has the prerequisite skills, having become adept at rapid programming for course projects. The major training issue will be in learning to understand and use the corporation's library of reusable components, as well as a specific tool set.

Basis for evaluation. Clearly, the key evaluation of the analyzer is whether closure was reached with the end users in a timely manner, and whether a complete requirements specifications was written that was understandable to the designer. The analysis prototyper has to create the executable prototypes quickly to obtain end user feedback. Because speed of completing each iteration is so critical, the prototyper is expected to write as little new code as possible, and instead to rely on retrieving and reusing software from the corporation's library of reusable components. Therefore, in conjunction with evaluating the effectiveness of the prototyper in completing each prototyping cycle in a timely manner, the prototyper is evaluated on his or her effective utilization of components in the corporation's library.

Title: Design Prototyper

Hiring manager. Company's key technical architect, perhaps the product technical strategist or key design architect for the project. Often, however, the design prototyper formally reports to the Product Development Manager or technical person responsible for overall management of a particular software project, and informally reports to the key design architect for the project.

Functional job description. Programs the executable prototypes during the design phase. Prototyper works with the designer to create an initial executable version of the proposed product. Should have some level of presentation or documentation skills to participate in the design review discussions.

To some extent, this person is the designer's pragmatic conscience, and is expected to give feedback to the designer as to whether the design ideas can be realized. As with the analysis prototyper, the design prototyper could be same person as the designer if the person doing the design is a competent programmer who finds it easier to think through a consistent design by describing the behaviors in a programming language.

Individual skills. Excellent programmer, who can create the design prototype. Also, an excellent explainer, who can work with the designer to prepare the architectural design document. The design prototyper is a conceptual

Object-Oriented Database System Manifesto

Condensed from an article by Tom Atwood in THE HOTLINE, January 1990

Rule 6: Thou shalt not bind prematurely

Dispatching of operations on objects, so that applications may manipulate a variety of object types, without having explicitly to name the operations on each type. This is a major tenet of object-oriented systems. Operation dispatching is described as "late binding" because the operation actually invoked is determined not at link-time but at run-time. In the absence of late binding, applications are littered with "case statements," which pair object types with operations, and must be updated whenever a new object type is defined.

Rule 7: Thou shalt be computationally complete

Any computable function must be expressible in the language. The rule is satisfied by essentially all programming languages, except for SQL, which is the popular relational query language.

Rule 8: Thou shalt be extensible

There shall be no distinction in usage between system-defined and user-defined types. Rather, all distinctions between them are masked by the implementation, so that they look the same from the point-of-view of the application.

DBMS GOLDEN RULES

Rule 9: Thou shalt remember thy data

OODBMS must provide for persistence of data, in the sense that persistent data continues to exist after the application creating the data has terminated. Moreover, persistence must be orthogonal to type so that any object may be stored persistently without special declarations or object translation. Finally, persistence must be implicit, ruling out explicit application commands to move or copy data in order to make the data persistent.

Rule 10: Thou shalt manage very large databases

Operations such as index management, data clustering, data buffering, access path selection, and query optimization are essential components of high-performance OODBMSs which are supporting large databases. This rule dictates that such operations be performed by the OODBMS, such that they are invisible to the application. There must be a clear separation between the logical and the physical level of the OODBMS.

Rule 11: Thou shalt accept concurrent users

One of the main virtues of DBMSs is that they allow multiple users to access the databases simultaneously, in such a way that they don't interfere with each other. The phrase "serializable access" is the database jargon for this kind of simultaneous access.

Rule 12: Thou shalt recover from hardware & software failures

This requirement is also a familiar one to those who have used traditional database systems. It must be possible to recreate any database in some coherent state, following a software or hardware, processor or disk failure. By "coherent state," we mean that the contents of the database after recovery reflect some single instant of time in the history of the database before the failure.

Rule 13: Thou shalt have a simple way of querying data

It must be possible to state database queries in a high-level and concise form, which is independent of application, and is susceptible to query optimization. The authors allow either programmatic interfaces or graphical interfaces to provide the query facility; the main point is that it be fast and easy to formulate and execute ad hoc queries efficiently.

"THE GOODIES"

The authors are aware that the Golden Rules will not provide an easy choice between OODBMS capabilities, because the Golden Rules are likely to be supported in some fashion by every offering. In contrast, the five optional features will provide more of a basis for comparing OODBMSs.

1: Multiple inheritance

The authors state that there is no firm consensus in the object-oriented community with regard to multiple inheritance between object types. Hence they regard support for multiple inheritance as being optional. We note that for systems that support multiple inheritance, such as C++ Release 2.0, it is possible to write applications using either single inheritance or multiple inheritance, while this is not possible using systems supporting only single inheritance.

2: Type checking and type inferencing

Although the authors classify compile-time type checking as an optional feature, they explicitly state that if more compile-time checking is provided, it is better. Type inferencing refers to the ability of the language compiler to deduce given declarations of base types. Sets of base-type objects represent conceptual derived types, based on how they are used. Type inferencing is very much a research topic, but the authors again say, the more the better.

3: Distribution

Any OODBMS may be distributed, in the sense that it provides simultaneous access to users running on different machines. There are many kinds of distribution, and some kinds of distribution can be viewed as a consequence of the concurrency requirement of Golden Rule 11. Nonetheless, the authors regard distribution as an optional feature.

4: Design transactions

For many OODBMS applications, the classical transaction model used for business DBMSs is not sufficient. These applications need some form of nonserializable "design transactions."

5: Versions

Hand-in-hand with design transactions is the ability to store efficiently and access various revisions of a single design. The authors applaud the utility of such a versioning capability, while noting that the capability is not in the set of core requirements.

PROJECT MANAGEMENT

Talking to project managers — project team job descriptions

Adele Goldberg & Kenny Rubin

Since the February 1990 issue of THE HOTLINE, we have been discussing issues critical to the successful introduction of object-oriented technology into large corporations.

We have directed our comments to project managers and discussed the impact that this new technology will have on an organization's project structures if the promised benefits are to become a reality.

In June, we sketched out a change in the project structure incorporating prototyping into the analysis and design phases of the software development lifecycle, and we emphasized the importance of formalizing the role of Reuse Librarian (or Reuse Manager). The purpose of this article is to respond to specific requests we have received asking us to provide job descriptions and evaluation criteria for each of the new positions that we have proposed.

Adele Goldberg is President and CEO of ParcPlace Systems. Goldberg spent fourteen years as researcher and manager at Xerox Palo Alto Research Center. She has served as president of the ACM and has received the ACM Systems Software Award.

Kenny Rubin is a computer scientist at ParcPlace Systems, where he has been a primary curriculum developer of three of the ParcPlace Systems Smalltalk-80 training courses and a lead instructor for many of the ParcPlace Systems courses taught over the past two years. Rubin has written on object-oriented technology, artificial intelligence, and human–computer interaction.

OBJECT-ORIENTED PROJECT STRUCTURE

There are two levels of software systems that might be built. One, the enterprise level, is a software system that captures the information models of an existing organization. The purpose of the system is to provide automation in support of carrying out the ongoing and well-understood activities of this organization. In such systems, the computer is used to handle the repetitive tasks and to provide safe long-term storage of the information related to carrying out these tasks.

Lower-level software systems assume that the enterprise level exists, but that additional tools are needed to facilitate carrying out specific activities of the enterprise. These lower-level systems are often process-oriented, specifying a particular usage scenario for the organization's information models. More often than not, the process requirements for these systems (i.e., how functionality is combined and accessed, such as a particular set of accounting tools) are not well understood, although the available information is already well specified, as it is part of the enterprise information model. Throughout our subsequent discussions, we assume that we are dealing with the development of the lower-level systems. We also assume that the desired process, and the form it will take as part of a software tool, are not well specified.

Let's review the main points we have made in past articles about the structure of a project and the composition of the project teams. The project structure incorporates prototyping to create an analysis proof-of-concept as well as a design architectural prototype. The analysis prototype is used to describe the intended resultant system and, specifically, what behavior that system will demonstrate. It is examined by the end users to determine whether all needed behaviors are captured in the proposed system, and to understand how these behaviors will appear in the final system. Once agreement is reached between the analysts and end users, a formal requirements specification is prepared. This specification is a contract for the subsequent development phases.

The design prototype is used by developers to test fundamental architecture and reviewed to determine whether the proposed architecture will support implementation of the required behaviors. The primary deliverable from the design prototyping process is a formal design document, provided to a separate product development team.

Typically, there are two distinct kinds of project teams in object-oriented development, only one of which is used on any given project: the Application Team and the Framework Team. The Application Team is responsible for the design, development, and delivery of a domain-specific application that fulfills the contractual obligation with a client. This team interacts with the client to determine the requirements, and focuses on reusing existing components to meet deadlines in a timely way. The productivity of an Application Team is tied to the usefulness of the organization's library of software components or assets.

The Framework Team has a fundamentally different charter than the Application Team. The Framework Team is concerned with the development of a basic general framework that can be customized to produce different end-user applications. Each of these applications is derived from the same general framework, but has domain-specific information and characteristics. To customize a framework in order to obtain a specific application means that the programmer selects optional existing components and then attaches these components to the framework. In addition, the programmer might refine (modify) the existing components, or create new ones, to include with the framework. An integral part of an application framework, then, is the library of components that can be used with that framework.

The result of the Framework Team's effort is a design that is reusable in other projects (unless the team reuses an existing design). Good designers create reusable, general frameworks that can be specialized. They also attempt to create basic components that fit into the framework, for example, representative data structures or information editors. Both the frameworks and basic components are stored in the organization's library of reusable software. The full project, then, benefits from both reuse and the creation of reusable components.

PROJECT TEAM MEMBERS

The Framework Team operates within six phases of the software lifecycle:

- Analysis prototyping — obtain the formal requirements specification.
- Design prototyping — demonstrate the feasibility of a particular implementation architecture.
- Reuse testing — test components to determine whether they can be reused to create a variety of applications based on the application framework.
- Reuse assessment — test the application framework and components to determine whether they can be reused for purposes outside the current project, and whether they are sufficiently general to warrant inclusion in the organization's library.
- Product implementation — product evolves to complete all functional requirements.
- Product testing — test suites exercise the prototypes and the product to determine whether all functionality works as specified.

In examining the responsibilities of development team members during these phases, we identify several roles that differ from those in a traditional development team. For the most part, existing team members can be retrained to fill these roles.

Some existing team members, such as the analyzer and designer, clearly have the same job in an object-oriented project or a non-object-oriented one. There is a key difference, however. This difference is that they must take an object-oriented point of view in their analysis and design. That is, they analyze from a behavioral perspective, and design with the goal of creating objects and object relationships that carry out the desired behaviors. Both object-oriented analyzers and designers understand the fundamental concepts of the object model, and the designers know how to translate the behavioral requirements into the particular details associated with the selected object-oriented technology. (Re)training is appropriate, in the sense that skilled (non-object-oriented) analyzers and designers, with the general knowledge associated with their jobs, will need to acquire in-depth understanding of the object model and its associated pragmatics.

> ***Retraining is appropriate, in that skilled, non-object-oriented analyzers and designers ... will need to acquire in-depth understanding of the object model and its associated pragmatics.***

The new members of the Team consist of:

- Analysis Prototyper
- Design Prototyper
- Reuse Evaluator
- Reuse Librarian or Manager

We now provide a description of each of these jobs. The format of each description is similar to that used by a human resources department when soliciting applicants for a particular job.

SPECIFIC JOB DESCRIPTIONS

Note that we write the job description for a single person who would fulfill the job. Depending on the size of the project, several people might be needed.

Title: Analysis Prototyper

Hiring manager. Product Development Manager or person responsible for overall management of a partic-

© 1990 ParcPlace Systems