Box 4
Folder 11

4 of 4

10273932

≡ 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'd like to see but don't — or whatever other feedback you'd like to send. Send comments to Robert Shelton, 1850 Union Street, Suite 1584, San Fransisco, CA 94123; Fax: (415)928-3036; 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	❑	❑	❑	❑	❑
Methods	❑	❑	❑	❑	❑
Components and Reuse	❑	❑	❑	❑	❑
Interview	❑	❑	❑	❑	❑
Book Review	❑	❑	❑	❑	❑
Product Announcements	❑	❑	❑	❑	❑
FYI	❑	❑	❑	❑	❑

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

VOLUME 3, NUMBER 8 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES JUNE 1992

Business in the Information Age

It seems that everywhere you turn today you can find articles and books heralding the dawn of a new era in human history: the Information Age.

John Slitz

The issue is not whether society is changing but rather what will the Information Age be and how will business organizations and their leaders adapt to meet its new demands? Alvin Tofler, an early and perceptive chronicler of societal change, offers the following statement in his latest book, POWERSHIFT,[1] to define the Information Age: "The most important economic development of our lifetime has been the rise of a new system for creating wealth, based no longer on muscle but on mind. Labor in the advanced economy no longer consists of working on 'things,' writes historian Mark Posner of USC Irvine, but of 'men and women working on other men and women, or … people acting on information and information acting on people.'"

Knowledge has always been associated with power; those who could obtain and use information effectively had power. The difference in the Information Age will come from the technologically enabled and enhanced distribution of information throughout a business enterprise and the speed with which that information can be turned into knowledge and action. As the twentieth century careens to a close, those organizations that focus on increasing their ability to turn information into knowledge will become the leaders of their industries. The issue for these new, world-leading companies is not whether to build pervasive, integrated information systems, but how to build them. The issue for Information Age CEOs will be how to plan, construct, and direct these new information-intensive organizations.

Another key issue to consider in the nascent information organization is the rate of information flow and the corresponding action that results. The tempo of our times is faster. We are driven by an increasing flow of information and the escalating need to act upon that information. This cycle of information and action is resulting in unprecedented levels of business, political, technological, and personal change. While change has always been with us, it was never before critically important to continually anticipate its effects in our business or personal planning. A few hundred years ago a

continued on page 4 …

IN THIS ISSUE

From the Editor

To remain competitive with Pacific Rim, Indian, Irish, and Soviet programming houses, US and European Community software development organizations have to change from one-off software engineering in the model of the job-shop to thoroughbred software manufacturing, engineering-to-order from off-the-shelf components—in the next three to five years.

This is no small challenge to a sheltered, specialized industry that has operated for several decades largely outside the scrutiny and performance demands routine in any Class-A MRP manufacturing organization. Furthermore, practitioners in this industry have yet to achieve a state-of-practice that would generally be viewed as engineering. Most commercial software houses and information technology (IT) organizations still do not discipline their members (technical or management) to use engineering methods and tools rigorously. We lament that our senior management does not see the value or deliver the committed support we require, yet the problem often starts with a CIO not providing executive-level linkage between the business and the technologists. A surprising number of organizations still balk—today—at spending a few thousand dollars on basic engineering tools for their analysts and developers. And, we are still talking about "the change problem" as though professionals in this industry have a personal prerogative to make software development an art form instead of a measured manufacturing discipline.

No matter. With the breakup of the USSR in the last year, a wealth of programming and engineering talent has become available—at pay rates that out-of-work junior programmers in the US would overlook even in these tough economic times. Indian programming job shops (sweat shops by Western standards) crank out code from specifications substantially cheaper than even domestic outsourcing firms. To boost their sagging economy, the Irish government has joined the fray, offering skilled technicians at labor rates 50% below those in the US, England, and Europe. Unlike the Indian developers, these competitors speak English and have a background in Western business practices. With the ascent to wealth of such formerly low-cost labor sources as Taiwan and South Korea, a third wave of Pacific Rim nations is following in their (and Japan's) footsteps as the brains-and-brawn-for-hire suppliers of the region. Programming muscle is available—cheap.

The pattern of the hard-goods manufacturing sector—moving rough component manufacturing, machining, subassembly, and even final assembly overseas in search of the lowest-cost labor source—is being repeated in software development. As the influx of work brings wealth, the standard of living rises; a commensurate rise in labor costs results, driving the work offshore, again, to the next low-cost labor source. Competitive advantage can be gained by the higher-cost supplier only by adding value in ways not accessible to the cheap-labor competitor. Failure of many a CIO to understand and act on this has helped fuel the current wave of outsourcing here in the US and, unless effectively countered, the trend will continue as globalization makes doing business across national borders easier through the end of this century and beyond.

At issue is not the sophistication of the work, fitness to specifications, technical documentation, or engineering discipline. The bottom line is cost. Cost is measured not only in actual cash outlay, but in terms of opportunity cost and market position. The offshore developer capable of delivering replacement core business systems for one of the largest global retail clothing businesses, headquartered in the San Francisco Bay Area, does not even employ programmers capable of commenting the program code in English—so they don't bother with internal documentation at all. The vendor does not agree with their customer that adherence to the customer's corporate data model is important, leaving this firm's data administration staff with serious integration problems that inhibit corporate-wide, not to mention interapplication, data sharing. (If you think this is just a data problem that will vanish with object

continued on page 3...

hotline on OBJECT-ORIENTED *technology*

The Hotline on Object-Oriented Technology (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012, (212)274-0640. Mailed First Class. Subscription rate — one 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 Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123. Voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of *Hotline on Object-Oriented Technology, Journal of Object-Oriented Programming, Object Magazine, The X Journal, The C++ Report, The Smalltalk Report,* and *The International OOP Directory.*

In the Industry

A five-year, $100 million program designed to increase US competitiveness in the semiconductor industry is entering the homestretch. Joint efforts among chip maker Texas Instruments, Inc. and two US defense agencies are expected to spawn revved-up chip plants based on single-wafer processes, distributed computing and object orientation by late next year. The team effort to hasten production turnaround time and gain real-time control of factory information in order to lower costs is expected to first be adopted by silicon makers and then spill over to other US businesses that are struggling to compete globally...The project seeks to accelerate chip-making cycle times using single-wafer processes—as opposed to traditional batch equipment—to quickly product low volumes of specialized chips...

Factory automation plan nears completion, Joanie M. Wexler, ***COMPUTERWORLD****, 4/6/92*

...PCWEEK: What is Cairo? [Microsoft Corp. Systems VP Paul] MARITZ: Cairo is the code name for a collection of projects we're working on that pertain to making [Windows] fundamentally easier to use in a distributed environment. It involves both putting object-oriented technology into the operating system, as well as getting that technology to work within a networked distributed environment. The vision we hold out is allowing PCs to become more like information appliances...

Great Expectations: Microsoft's plans for Windows, Paul M. Shere, ***PC WEEK****, 4/6/92*

...The International Business Machines Corporation, the world's largest computer maker, asked Sun in February to join the Taligent joint venture that it formed last year with Apple Computer Inc., the second-largest maker of personal computers. The request to Sun was an attempt by IBM to counter Microsoft's introduction later this year of soft-ware for work stations, the powerful desktop computers increasingly popular with corporations. IBM's overtures were confirmed yesterday by industry executives who did not want their names publicly attached to such preliminary discussions. Sun, which has been left out of such alliances or shunned them, and IBM would not comment...Evidence that Sun has become more receptive to cooperative projects has come in recent months, with the company forming several partnerships with other companies to work on specific technologies. It recently took on the little-noticed Object Management Group partnership with the Hewlett-Packard Company aimed at setting Unix software standards...

Sun link is sought by IBM, John Markoff, ***The New York Times****, 3/13/92*

...For [BT North America's Mike] Roberts, C++ has proven to be a boon to efficiency and productivity. Originally a C shop with an emphasis on real-time programming or as Roberts explains, "extreme real time—a step beyond what people think of when they say real time. Consequently we have to be able to get down and play with the bitsies when we need to." Roberts' team finds C++ cleaner than C and easy to use when managing a project. Given the turnover rate at most organizations, assigning a half-completed project to a new programmer is often a nightmare. According to Roberts, "C++ is much easier to explain than conventional code. You can see how the objects break the problem into understandable pieces. For that reason it is easier to maintain."...

A plus for C++, Jessica Keyes, ***PCAI****, 4/92*

...The [Distributed Management Environment] breaks new ground in software development because it utilizes advanced object-oriented software technology. OSF argues strongly that this powerful new approach to structuring software will solve the complex distributed systems management problems that users say are their primary challenge today...

Bringing the DME into sharper focus, James Herman, ***NETWORK WORLD****, 3/30/92*

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: Robert Shelton, 1850 Union Street, Suite 1584, San Francisco, CA 94123; fax: (415) 928-3036.

June 8–12, 1992 **USENIX** San Antonio, TX Contact: 614.588.8649	June 15–19, 1992 **Xhibition '92** San Jose, CA Contact: 617.621.0060	June 16–18, 1992 **SD '92** London, UK Contact: 081.742.2828	July 14–17, 1992 **Object Expo Europe** London, UK Contact: 212.274.0640	July 21–23, 1992 **Object World** San Francisco, CA Contact: 508.879.6700

F.Y.I.

Excerpts from leading industry publications on aspects of object technology

Engineering Methods

...Taking the object perspective during analysis, design and programming mandates special talents, skills and genius. Data flow techniques—the current rag—only require that observers act as bloodhounds who trace the flow of paper. Where objects are the goal, "Many are called, but few are chosen."...For too long, we've believed that endless process, protocols and software packages could do our analysis, design and programming—and we probably always will. The OOP approach forces the industry to recognize that we have too many people, inappropriately selected and trained, with the wrong set of tools. Will we ever learn?

Letters: Where many are called and few chosen, Howard D. Weiner, ***COMPUTERWORLD****, 3/16/92*

...Software object is not another expression for program module. However, many software professionals have difficulty discerning the difference between simple modularity and designing with software objects...A [State Transition Diagram] is a good technique to determine whether you're really designing a software object. If you can represent the object with a discrete set state changes you probably have an object. Frequently, an object can be further decomposed into other objects.

Objectively speaking: Objects in design, Richard Riehle, ***HP PROFESSIONAL****, 3/92*

Multimedia

...In addition, while high-power RISC technology is being built into PCs, video hardware such as computer-based multimedia machines are expected to become more sophisticated. Giant consumer-electronics companies like Sony Corp. are threatening to dominate the home market at the expense of multimedia PCs. Some observers believe that the motivation behind recent technology agreements between erstwhile rivals IBM and Apple has less to do with the onslaught of cheap computer clones than with trying to establish a platform to compete with the likes of Sony later in the decade...

High Power: Next generation PCs, Michael Antonoff, ***POPULAR SCIENCE****, 4/92*

Standards

...Typically, in the computer industry, many different vendors introduce different and incompatible versions of a new technology. Proprietary market wars persist for years and sometimes decades, creating a heterogeneous mess in which users fund it impossible to integrate systems. In the emerging object-oriented Enabler environments, this shakeout has substantially occurred before there were many products...

Advances and Research: The future of manufacturing Enablers, James E. Heaton, ***ESD TECHNOLOGY****, 3/92*

...Thus the war over whose objects to use is likely to replace the war over operating systems. And the lack of a common foundation on which to build the software that enables computers to do increasingly sophisticated tasks is likely to continue to inhibit the American computer industry. "It seems almost certain now that the battle lines are drawn," said Andrew Singer, a computer researcher at Enginuity, a high technology company based in Palo Alto, CA.

Ideas & Trends: New weapons prolong the computer wars, John Markoff, ***THE NEW YORK TIMES****, 4/5/92*

...If you're inclined to believe that what the world really needs now is another UNIX Standard, you'll be delighted to learn that Novell is developing an object-oriented version that will, of course, be called Univell. Think of it as Novell's own self-contained Taligent project. In any event, sources tell the Knife that the theme of the project is to develop a server-oriented environment for the networks of tomorrow...

IBM must WIP it real good, Sergei Krikalev, ***MACWEEK****, 3/30/92*

Languages

...[David] Taylor and other proponents of object-oriented programming stressed one point as highly significant: Using object-oriented languages such as C++ and Smalltalk does not guarantee that software is object-oriented. These languages are simply programming languages set up to make the mechanisms of object-oriented programming easy to execute. Procedural programs can be written in C++ and Smalltalk, while object-oriented programming can be done in procedural languages. Indeed, even simple languages like BASIC and Pascal have been given object-oriented extensions...

Taking the "lego" approach to software design, Mitch Wagner, ***OPEN SYSTEMS TODAY****, 3/16/92*

...The interactive engine allows programmers to execute software one small section at a time, said [CenterLine's president and CEO Sesha] Pratap. "It's vital for effective object-oriented programming, because you want to be able to write several objects and execute them, and build your program by constructing objects and linking them together one at a time," he said. "This is, in fact, how most advanced programming environments are designed. It's one of the reasons why LISP was able to yield such huge productivity gains."...

Coping with the OOP challenge, ***OPEN SYSTEMS TODAY****, 3/16/92*

...continued from page 2

technology, consider again: what would happen to your reuse strategy if an offshore developer did not respect your corporate class model?)

Yet for this multibillion dollar company, the savings in development costs alone make a persuasive business case. Add to this the speed at which applications can be coded offshore (about four months each in this case), and we're talking about being another inventory turn ahead of the competition in the rough-and-tumble rag trade—a business in which getting ahead depends on divining subtle customer preferences (and changes thereto) that the customers cannot even articulate (hmmm... sound familiar, software developers?), and staying ahead depends on changing your offerings and turning over inventory faster than the other chain. This company's senior management is willing to live with what they perceive as the costs of this style of software development because the measurable costs are significantly less than the costs of known problems with today's systems—and incalculably less than the cost of being second.

Approaching software engineering discipline, CASE, object technology, or any other technical innovation, for that matter, from the perspective of "look at the technical problems we solve with this, and the technical benefits we can reap" is about as effective as communicating in a whisper on a foundry floor—the background noise simply drowns the whisper by several orders of magnitude! Selling object technology on the strength of claims of 14:1 productivity improvement and software concepts that model the real world, thus eliminating the analysis problems we have experienced with existing technology, is whispering into the background roar. Our analysis problems have been largely people problems. The productivity gains are to those disciplined enough to buy, reuse, adapt, adjust—all those things that run orthogonal to the grind-your-own-concrete-mix training of software *developers*. To the flexible go the spoils.

Coming from very different perspectives and backgrounds, our authors in this issue approach object technology with these themes in common: software manufacturing, innovation and process improvement, and adaptation to market changes.

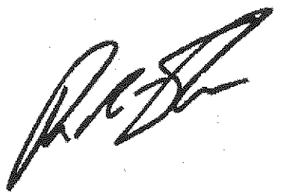
Mr. John Slitz of the Object Management Group addresses innovation and information technology in the context of historical changes in Western society over the last several hundred years. Business competitiveness today, he posits, is driven by information access and use, on "the ability to gather and process large amounts of data, and deliver it into the hands of well-trained workers capable and responsible for making timely decisions...." The competitive business will have a flatter structure than today's military-styled hierarchies—more like a peer-to-peer LAN than an SNA network (for the hard-core mainframers among us!). Value added will be measured in knowledge-per-unit instead of cost-per-unit, with the business impact of knowledge being to increase the revenues per employee and drive down the incremental cost of each new revenue dollar. Those rising to this challenge will be organizations able to change—in his use of the term, to innovate. Yet Mr. Slitz's concept of innovation carries strong overtones of both the management of dramatic change and the institutionalization of business process improvement. We'll revisit this in our interview with Mr. Merlyn...in a moment.

Mr. Robert Brown of the DataBase Design Group examines a migration path from today's state-of-practice semantic modeling to object modeling. The objective is to develop more effective tools for understanding our businesses—while building on foundations already in place in many organizations. These modeling tools fill a crucial role in the move to software manufacturing: once we can manifest and understand—really blueprint—a business, we can make effective comparisons to off-the-shelf components. To borrow an analogy from noted author (and HOTLINE contributor) Mr. David Taylor, this is similar to the way we would build a new PC today: lay out the component schematic, then fill it with chips and assemblies purchased from parts catalogs. Those who suggest that this modeling is superfluous, intellectual space filler have yet to spend enough time with their business partners to recognize that comparing code to business activities in not an effective mechanism for understanding either the business or the code!

Mr. Mark Anders of Inmark Development joins us to write about frameworks—prefabricated application building blocks that are a good example of what Mr. Brad Cox calls the software-IC. Frameworks are among the components that can be selected from today's software "parts catalog" for developers using C++, Smalltalk, Actor, and even some nonobject languages. Mr. Anders presents the developer's perspective on the importance of using such off-the-shelf components, giving us some hard numbers that, as development managers will appreciate, translate directly into development and maintenance cost savings.

We interview **Mr. Vaughan Merlyn** of Ernst & Young, wherein he addresses the conflict in Western culture between innovation and continuous process improvement. Success with business process innovation and business reengineering depends on first constructing the foundation: an organization that can support change. This Mr. Merlyn terms *the learning organization*. He views innovation and process improvement as two distinct issues, and feels our Western values are causing innovation strategies like business reengineering to overshadow the more fundamental issues of change embodied in continuous improvement. Continuous improvement, after all, is the ability to sustain the value of innovations—for us software developers, it's called "maintenance and enhancement," while innovation is called "development." Those unable to implement such concepts as Total Quality Management will not be able to manage the more abrupt changes of reengineering. Those who don't ultimately achieve both will be surpassed by other market players who are able to change. Like Mr. Slitz, Mr. Merlyn also forsees responsibility-driven, flat organizations where change is the norm. He also gives specific suggestions on what needs to change in our organizations to achieve the leverage of the learning organization.

Central among these changes for software development organizations is to understand software development as manufacturing. The successful software developers of the next three to five years—those able to compete against offshore vendors—will be those who make the change from today's one-off craftworks to manufacturing from commercial components. Object technology is one technology enabler. What we practitioners and our business partners consider to be acceptable attitudes toward software development is another. If we choose to not decide, our offshore competition will help our customers choose for us. ≡≡



... continued from page 1

farmer expected to live his entire life with the techniques his father taught him and to pass on those same techniques to his son. Tradition, down to the smallest detail, dictated both correct behavior and expectations. And traditions changed slowly, over generations.

That situation only changed in the nineteenth century and was buried in the twentieth, with the advent of the Industrial Age. In this period, wholesale movement from farms to factories produced "wonders" of science that resulted in the creation of our modern world. With the advent of fast ships, airplanes, telephones, and then the computer, the pace of information flow, life, and change have accelerated to the point where the focus of successful businesses in the 21st century will be primarily concerned with channeling and using information.

The movement of society from Agricultural to Industrial to Information eras may be viewed as the history of information processing. With each successive epoch the speed of dissemination and the value and pervasiveness of information has increased dramatically. Each epoch has had to develop new means of production, management, and political institutions to cope with the issues and challenges of the time.

PRE-INFORMATION AGE HISTORY

The past can be viewed in three broad ages: Agricultural, Industrial, and the current beginning of the Information Age. In earlier periods, the majority of the population was involved in either agricultural or industrial activity. In the Agricultural Age, a majority of the population worked at agricultural tasks, producing food or other materials directly from the land. Success was achieved by increasing the yield per parcel of land and those landowners who were the best producers were considered the most successful. Movement was difficult and the diffusion of "technical" information—on crop production, harvesting, or irrigation—was limited to the pace at which a man could walk. It was a safe bet that most people were born, lived, and died all within a few miles. This meant that the world looked almost the same from decade to decade. The pace of life was slow and change played almost no part in anyone's planning.

The next period was exemplified by the Industrial Revolution and started in the mid-1700s. The world's information creation and transmission abilities speeded up. This resulted not only in an increased ability to produce manufactured goods, but in increased production from agriculture as well. Society as a whole gained the ability to learn and change. Knowledge itself was beginning to be viewed as having some value, although not much. Rather than yield per acre the predominant measure became cost per unit. The method of accomplishing lower costs was through "economies of scale." Essentially, bigger shops and then bigger factories discovered they could produce more goods at ever lower costs per unit.

If we focus on the speed of communications as a key indicator of the pace of societal change, we see the last 200 years as a continual increase in speed marked by higher levels of productivity in each area of the economy. It would seem that the ability to move and share information is a prime driver of technological and societal progress. It is noteworthy that we still produce agricultural product in vast amounts, but use only a small fraction of today's population to produce more then the entire population could produce 200 years ago.

Just as the Agricultural Age valued yield per acre and the Industrial Age relentlessly tracked and recorded cost per unit, it is my belief that the predominant measure of the Information Age will be knowledge per unit. The labor and material costs of each unit produced from the manufacturing sector will continue to decrease but a new goal to increase the value of each unit produced will be added. This new measurement will allow products—whether service, leisure, intellectual, agricultural, or manufactured—to be comparatively valued. It is also my contention that those companies that seek to maximize knowledge per unit in their products will come to dominate the Information Age.

The Industrial process sought to use the "magic" of mass production, the learning curve, and large, capital-intensive plants to bring down the per-unit cost of the goods produced (Fig. 1). They also used information in the guise of manufacturing "know how" or in the slow (by today's terms) movement of knowledge from the research laboratory to the industrial shop floor. The reason large enterprises could be built was at least partly due to the relatively slow pace of change in the economy. A plant then took ten years to plan, design, build, and begin to produce a payback. This required a low cost of capital and the assurance that in those ten years the product produced would still be competitive, if not dominant in the market. It also required that a huge, undifferentiated market be available to consume the prod-

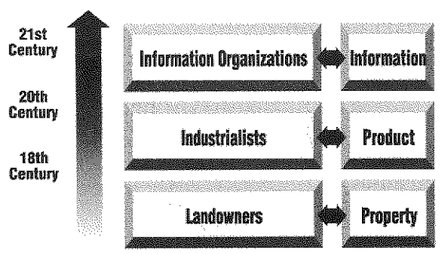


Figure 1. Industrial Revolution Chart

Partnerings & Acquisitions

William P. Lyons, president and CEO of the **Ashton-Tate Corporation** before its acquisition by **Borland International, Inc.** last year, succeeds Adele Goldberg as president and chief executive of **ParcPlace Systems.** Goldberg continues as chairwoman.

Dr. L. Peter Deutsch joined **Sun Microsystems Laboratories,** Inc. (SMLI), the advanced research subsidiary of Sun Microsystems, Inc., He has also been named a Sun Fellow, one of three within Sun Microsystems. At SMLI, he is helping plan future software development technologies and software architectures. Most recently, Dr. Deutsch spent five years as chief scientist at **ParcPlace Systems**. He also spent more than 14 years with **Xerox** at the Palo Alto Research Center.

David E. Liddle, formerly a research scientist at **Xerox Corp.**'s Palo Alto Research Center and founder and CEO of **Metaphor Computer Systems,** and Paul G. Allen, cofounder of **Microsoft** and founder of **Asymetrix Corp.**, formed **Interval Research Corp**. Liddle was named president and CEO, and Allen will serve as chairman and provide funding for the company. Interval's goal is to perform research and advanced development in information systems, communications, and computer science. Specific topics for research have not been announced.

Borland International Inc. reached an agreement to acquire two professional programming tools from the **Solution Systems' division of Software Developer's Company, Inc**. Under the agreement, Borland would own, develop and market BRIEF, a professional programmer's editor, and SOURCERER'S APPRENTICE, a network version control system that manages the building of large software projects. Under the terms of the agreement, closing of the transaction is subject to certain contingencies.

Versant Object Technology Corporation signed a licensing agreement with **IBM** relating to the VERSANT Object Database Management System (OBDMS). Under this agreement, IBM and Versant will work together on technology for a future IBM AIX CASE integration framework offering. Additional terms of the agreement were not disclosed.

Solbourne Computer Inc. set up a separate software business unit to sell and license graphical user interface tools enabling the develpment of C++ applications running on Sparc-compatible systems. Previously, the tools were only available bundled with Solbourne hardware. Solbourne's first licensees include **CenterLine Software Inc., ParcPlace Systems**, and **The Qualix Group**.

The Object Management Group (OMG) and **World Expo Corporation** announced the 2nd annual Object World Exposition and Conference, to be held July 20-23, 1992 at the Moscone Convention Center in San Francisco, California. The OMG and **Computerworld** will sponsor the First Annual Computerworld Object Application Award in conjuction with the show.

UNIX Systems Laboratories and **Tivoli Systems** recently agreed to develop a unified object-oriented framework for UNIX System V.4. The technology will ease the creation and management of distributed computing environments. The resulting product will be based on Tivoli's WizDom object environment.

A joint development and technology sharing agreement calls for **Tivoli Systems Inc**. and **Sun Microsystems, Inc**. to work with the **OMG** to create UNIX-based system managment software for managing distributed networks. The jointly developed tools will be bundled into a future version of Sun's Solaris operating system.

Object Design was designated by International Business Machines Corporation as an **IBM** Business Partner. IBM also is the largest user of Object Design's ObjectStore ODBMS. IBM's Technology Products group is using ObjectStore in its internal Electronic Design Automation (EDA) group to build software tools for designing systems and components for future RISC System/6000 models. Object Design and IBM will work together to jointly market ObjectStore on IBM's RISC System/6000 family of advanced workstations and servers. In addition, the firms signed a joint technology development agreement to work together to provide early availability of ObjectStore on new releases of IBM's workstations and servers products.

Object Design also announced the signing of a value-added reseller (VAR) agreement with **MediaShare Corporation** of Carlsbad, CA. Under the agreement, MediaShare has licensed the ObjectStore for Windows ODBMS and will ship an embedded runtime version of ObjectStore in its forthcoming PRISM, an interactive sales and marketing product information system.

Object Design entered into a strategic alliance agreement with **Fluent, Inc.** of Natick, MA, to integrate its object-oriented database management technology with Fluent's digital video and audio software products for multimedia application development.

ExperTelligence, Inc.,
5638 Hollister Ave., Suite 302, Goleta, CA 93117, 805.967.1797

ExperTelligence announced the release of **The ExperDocumenter** on Action! for Digitalk's Smalltalk/V. The ExperDocumenter scans the source program for the "classes" selected by the user and automatically generates the description,classes, methods, arguments, and all technical information needed by the programmers and certain categories of end users. The output of the Documenter is a Rich Text Format (RTF) file directly readable by any word processor that uses this format.

Computer Innovations, Inc.,
980 Shrewsbury Ave., Tinton Falls, NJ 07724 908.542.5920

Computer Innovations announced two programs for the Microsoft Windows NT environment: **DEBUG*2000**, a source code debugger, and **EDIT*2000**, a programmer's editor. DEBUG*2000 is a source-level debugger for C/C++ programs that minimizes typing by using windows that automatically update as a program runs and emphasizes efficient use of screen space. EDIT*2000 is a programmer's editor that provides a full windowing environment. Support of the mouse, pull down menus and an effective Help system makes EDIT*2000 easy to learn and fast to use. A programmer may open as many files as desired and editing becomes fast and easy through the use of simple mouse clicks, keystroke commands, or any combination that suits the programmer's style. DEBUG*2000 and EDIT*2000 both run under Windows NT and Windows 3 on Intel 386/486 platforms and Windows NT on the MIPS platform.

Hewlett-Packard,
3000 Hanover St., Palo Alto, CA 94303 408.720.3632

Hewlett-Packard Company is shipping **HP NewWave 4.0 Desktop Manager** software for Microsoft Windows. A major upgrade from Version 3.0, HP NewWave 4.0 offers significant improvements in all areas of desktop management including the desktop organizer, workgroup library, and work-automation support through its exclusive agent macro facility and OLE 1.0 support. Among the improvements to HP NewWave's desktop organizer are drag-and-drop printing, 32-character titles for all data files and drag-and-drop file attachment of Windows documents to the NewWave desktop, as well as previously introduced features such as icons, folders, and file drawers. Now all DOS and Windows applications are interoperable with the NewWave desktop manager. The enhanced software installation process makes DOS and Windows applications available as icons on the desktop automatically.

Versant Object Technology,
4500 Bohannon Dr., Menlo Park, CA 94025, 415.329.7500

Versant Object Technology announced the availability of the VERSANT **Interactive C++ Tool Set,** a C++ software system for developing graphical end-user business applications in object-oriented environments. The Tool Set is designed to support a new class of applications including multimedia, workgroup, and second-generation client/server applications. The VERSANT Interactive C++ Tool Set is tightly intergrated with the VERSANT ODBMS giving users a seamless development environment for building and deploying object-oriented database applications. The Tool Set consists of VERSANT Screen, a workstation-based tool that allows users to create highly interactive applications; VERSANT Interactive Object SQL, which uses SQL like statements to access and manipulate objects within the VERSANT ODBMS; and VERSANT Report, an interactive report generator.

Lenel Systems International,
19 Tobey Village Office Park, Pittsford, NY 14534, 716.248.9720

Lenel Systems International is currently shipping **MediaOrganizer**, multimedia object management software for Microsoft Windows users that is Windows 3.0 and 3.1 compatible. With MediaOrganizer, users can quickly organize, catalog, retrieve, and play all types of multimedia information including full-motion video, digital and analog audio, graphics, animation files, still images, and text. Users can also create multiple, scalable windows for simultaneous display of their multimedia information. MediaOrganizer meets the challenge of organizing, cataloging, and retrieving large amounts of multimedia information by integrating three critical technologies: multimedia computing, object-oriented programming, and database management in a Windows environment. This tightly integrated software will also serve as the foundation for a complete family of upcoming Lenel products including presentation, editing, authoring, etc.

Inmark Development Corporation,
2065 Landings Dr., Mountain View, CA, 415.691.9000

Immark Development Corporation announced the release of its application framework, **zApp**, for the MS DOS platform. zApp for DOS will offer DOS applications the same look and feel as MS Windows applications, including Windows style pull-down menus, dialogs, scroll bars, and controls. zApp for DOS supports Borland, Zortech, and Microsoft C++ compilers. It offers single source-code compatility with zApp for Windows, as well as data entry forms, Transparent MDI Support for DOS text mode, frame/pane architecture, and automatic window sizing and positioning. Other features included are logical size and dimensioning of objects and flexible message and background tasks

EMS Professional Shareware Libraries,
4505 Buckhurst Ct., Olney, MD 20832, 301.924.3594

EMS Professional Shareware Libraries began shipping an enhanced new version of its **C++ Utility Library.** The new version adds 18 new Public Domain and Shareware C++ language products, making a total of 140 products for professional C++ programmers. All products in the library (and all known commercial C++ PC products) are described and indexed in a database that accompanies the library.

ucts in numbers sufficient to justify their production. To an ever increasing degree, these conditions are disappearing. Capital is no longer cheap, markets are becoming more fragmented, and the lifecycle of many products, even the most staid, is measured in months and years, not decades.

CEO TO INFORMATION MASTER?

In the Information Age, companies will have to react faster and produce a broader range of products better suited to the ever changing tastes of more sophisticated and differentiated consumers. To accomplish this task, successful companies, in the motto of Software AG, a large mainframe software developer, must "listen hard and respond fast" to the changing market condition, as well as to the changing technological base for their product and the shifting demands of their customers. This process of constantly monitoring and analyzing their own products and processes, and then making the necessary improvements, is called *innovation*. Successful enterprises must become innovations masters, which will require new methods of planning, measuring, and managing their businesses.

The ability to gather and process a large amount of data and deliver it into the hands of well-trained workers capable of and responsible for making timely decisions will be critical. Success will be judged on the knowledge that can be brought to bear on each unit produced by the enterprise. The higher the knowledge content, the higher the value to purchasers of the product. Increasing the knowledge content in each unit of production must be the goal and scorecard of top management and the means to this end is through innovation planning and the development of pervasive company information systems to power this information flow.

All businesses are not going to experience change at the rate of computer chip manufacturers, but the leaders in every industry must come to grips with a pace and scope of change and a need for product innovation that is far greater then they have experienced in the past. This means that top management must be involved in designing and implementing the information systems that will become the nervous system of the new corporation.

The Information Age CEO can no longer leave the issues of information flow to the IT manager. Every major area of the company must be evaluated on its contribution to the knowledge content/value added to the enterprise's product. Each employee must be trained on the goals, market, product and information technology used to "plug" him/her into the company. Tasks in the company must be analyzed on their past, present, and future contributions to the value of the company's products. A strong link between the CEO's vision, the company's strategic plan, and the IT plan must be forged. The Information Age company cannot afford to have its information systems strategy out of step with the business' strategic plan.

> ***Successful enterprises must become innovations masters.***

Each major function of the company should be given an innovation target (see Fig. 2) so that progress and contribution to the company's goals can be assessed. An optimal rate of innovation and the knowledge value contributions from that rate or higher investment should be tracked and evaluated. If the rate of current or expected innovation is low it may mean that the function is a well-understood process and therefore should be marked for increased automation, out sourcing, or being left alone.

In those processes where innovation is still occurring, often called state of the art, effort should be made to increase the pace of innovation until the rate of change stabilizes and the process can be shifted into the company base. In those areas where significant innovation will lead to a disproportionately high gain in the knowledge content of the product, top management should be actively involved. Likely areas for such involvement are in the transition of information systems to speed the flow of analysis information, new sales channels or methods, design changes that better fit customer input, and market analysis to anticipate changes in demand. At each level of the chart, the power to implement procedure-changing decisions must be given to the line employees best positioned to make those changes.

Incremental improvements must be an accepted part

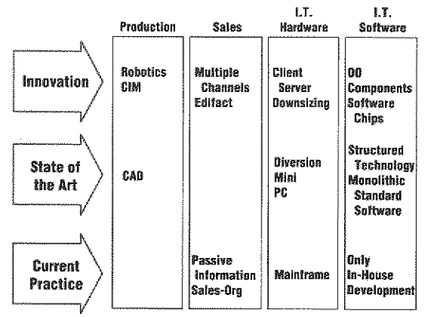


Figure 2. Technology decisions.

of the each function and the responsibility of the personnel within that group. Such personnel must also have both the authority and the responsibility to implement innovation. Innovations accumulated throughout the organization by these semiindependent, functional groups will lead to improvements in the entire system. By training each employee across a broader area of the company, not just job-specific tasks but those of related functions as well, greater autonomy and intelligence can be brought to bear on all problems.

THE OBJECT-ORIENTED ENTERPRISE

To create an Information Age enterprise, management must focus on the flow of information, as well as the development and communication of a strategic vision for the company. In implementation, each business function can be modeled as an object in the information system. Communications between functions can be modeled as messages. Within each function, the combination of people, information, and structure that provides a specific and measurable increase in value to the organizations product can also be modeled as an object. By adopting an object-oriented perspective in both strategic and information system planning for the enterprise, the tendency for the IT and strategic plans to disconnect can be eliminated.

Top management derived its name because it resided at the "top" of a hierarchical organization. In Industrial Age enterprises, information flowed up the organization and only those at the top had all the information necessary to make decisions, which were then communicated down to the organization. With the new information technology, a company's information will be available to every employee (with the exception of confidential proprietary patent, trademark, specific R&D, personnel, and salary data). This will engage more people in the decision-making process and lead to faster response and better products.

Management will no longer be at the top of the organization issuing orders to a passive workforce. Instead, key executives will be the architects and facilitators of a corporate vision and the designers of a responsive Information Age organization. Management will set goals in support of the company vision. Decisions on tactical and implementation levels must be made in each group without direct involvement by management. Employees with access to all the company's information can produce the rapid innovations and quality improvements the information enterprise needs to prosper in the tough competitive market of the Information Age (see Fig. 3).

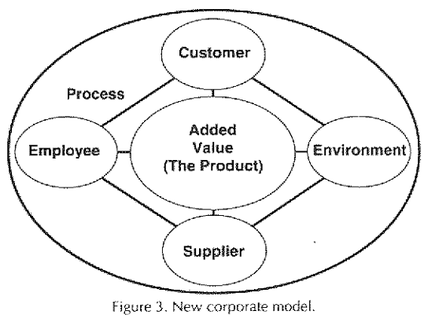


Figure 3. New corporate model.

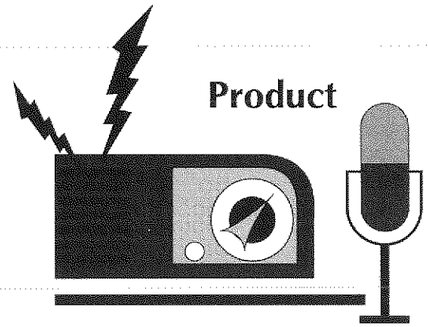
CONCLUSIONS

As the Information Age dawns, the demands on management will stress individuals and organizations to the limit. Constantly shifting markets, technological advances shortening product lifecycles, and a more specialized and independent work force are a few of the better-understood challenges that lie ahead. It is my hope that by offering the perspective of the enterprise as an information processor, rather than a material or people processor, new insights may be gained. The mechanisms to help plan and manage Information Age organizations are not fully developed, but I believe that a link between the strategic-planning/vision-creation function of top management and the information systems implementation must be forged. Information processing is on the way to adopting object technology as a new paradigm for the construction of diverse, networked, multivendor systems. Because object technology seeks to model real-world "objects" as persistent software entities and then build new systems by reusing and manipulating them within the computer, a necessary link to the business plan is possible. Much work lies ahead, but the integration of the information system of an enterprise with the strategic vision of top management is the primary requirement of the Information Age enterprise. ≡≡

References and suggested reading

1. Tofler, A. Powershift, Bantam Books, New York, 1990.
2. Rothschild, M. Bionomics, The Inevitability of Capitalism, Henry Holt Publishing, 1990.
3. Peters, T. Thriving On Chaos, Alfred A. Knopf, New York, 1987.
4. Sakaiya, K. The Knowledge Value Revolution, Kodansha, Japan, 1985.

John Slitz is Vice President of Marketing for the Object Management Group, a nonprofit, member-supported corporation of computer vendors, software suppliers, and IT users. Before joining the OMG, Mr. Slitz was a founder and Vice President of Strategic Development for Netwise, Inc. He holds a BA in Economics from SUNY Cortland, an Executive MBA from Fairleigh Dickenson, and a MA in Psychology from The Graduate Faculty of the New School for Social Research.



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.

NeXT Computer, Inc.,
900 Chesapeake Dr., Redwood City, CA 94063, 415.366.0900

NeXT Computer began shipments of its **NeXTcube Turbo workstation.** The new workstation is built around the Motorola 68040 processor running at 33 MHz and includes two newly designed NeXT custom chips. The new Turbo machines are rated at 25 MIPS. At the same time, NeXT reduced the prices on its most popular configurations by an average of 10% compared to last year's configurations. NeXTcube Turbo pricing is unchanged from current NeXTcube pricing. Five configurations of NeXTcube Turbo are now available and can be configured with up to 32 MB of DRAM (they will be able to support 128 MB when 16 MB parts are available) and 2.8Gb of hard disk drive storage space. The NeXTcube Turbo also supports three NeXTbus expansion slots providing expansion flexibility for such add-in products as NeXT's NeXT dimension 32-bit-per-pixel color/video board.

Autumn Hill Software,
145 Ithaca Dr., Boulder, CO 80303 303.494-8865

Autumn Hill released the next generation of its Menuet GUI, completely rewritten in C++ and now providing extensions for pen-based computers. **Menuet III/CPP** specifically supports Communication Intelligence Corporation's PenDOS operating environment. The DOS version of Menuet III/CPP features a generic graphics API that interfaces to a wide variety of commercially available graphics libraries. The product is also available in versions for various 16-bit and 32-bit DOS Extender environments. A Windows version of the product will be available by the end of the second quarter of 1992, and versions are being planned for IBM/s OS/2 2.0 and SunSoft's Solaris. All major C++ compilers are supported. Menuet III/CPP provides a Motif-style look and is fully CUA compliant.

Borland International, Inc.,
1800 Green Hills Rd., P.O. Box 660001, Scotts Valley, CA 95067-0001 408.439.1631

Borland International Inc. introduced a combination software and video training product, **"Learn Programming Today,"** designed for novice computer programmers and power users who want to learn to program with Borland's object-oriented Turbo Pascal 6.0 compiler. Also included are 50 sample computer programs, a two-hour videotape (11 lessons), and a 350-page workbook. The videotape features Zack Urlocker, Borland's Turbo Pascal product manager, and the workbook was written by Keith Weiskamp, a popular computer book author. Borland also introduced two new video training courses for use with the company's object-oriented Windows programming software. **"World of ObjectWindows for C++"** and **"World of ObjectWindows for Turbo Pascal"** are designed to teach advanced programmers how to create Window applications quickly and easily using Borland's ObjectWindows software. Each ObjectWindows training course includes two one-hour videotapes and a workbook with examples. Complete source code is also included, enabling users to work with the lessons as they appear or customize the lessons for personal applications. The videotapes feature David Intersimone, Borland's Director of Developer Relations and the workbooks were written by noted computer author Tom Swan.

The Stepstone Corp.,
75 Glen Rd., Sandy Hook, CT 06482, 203.426.1875

Stepstone announced **Objective-C** for the Macintosh. Objective C offers both dynamic and static binding. The ability to use ANSI C is retained at all times. Stepstone is also shipping V4.2 of the Objective-C Compiler with ICpak 101, the Foundation Class Library. Twenty classes and more than 300 methods provide a base of reusable components to build custom classes. The Objective-C Compiler is a preprocessor that requires MPW C and is compatible with the MPW Development Environment. A Macintosh computer with a hard disk and 4 MB of RAM (8 MB for larger applications) is required. It ships with a full set of documentation and tutorial.

The Stepstone Corp also announced the availability of Objective-C and its GUI toolkit, ICpak 201, for the MIPS Magnum 3000.

Arbor Intelligent Systems, Inc.,
506 N. State St., Ann Arbor, MI 48104 313.996.4238

Arbor now offers an MS-Windows version of its bridge from Objectworks Smalltalk to Nexpert Object, an expert system shell from Neuron Data. The Smalltalk Nexpert **OBJECT BRIDGE** enables developers to take a knowledge base created with Nexpert and embed it along with Nexpert's inference engine in a Smalltalk application. All objects created in Nexpert thus become available as Smalltalk objects. Expert systems created with this bridge are instantly portable across Macintosh, MS-Windows, and UNIX X-Windows without change to the code.

Send Product Announcements to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123, fax: (415) 928-3036. Include company name, address, and phone number.

didate classes?! Perhaps this is a reflection of the underlying, predominantly bottom-up approach seen elsewhere (e.g., p. 44).

The authors' definition of the association relationship appear to be nonstandard. They conceive of an association relationship as being when "one entity is a container for other entities" (e.g., p. 95). Association is, I believe (see also Rumbaugh [2]) much wider than just containment. Container classes are related more to the concept of genericity. A second important, and not yet fully defined, relationship is that of aggregation. Whilst appreciating that there still remains work to be done in clarification of such object relationships, I find the discussion on page 47 of the is-part-of relationship totally unconvincing. Finally, the idea of creating two classes (one abstract and one concrete) for any brand-new class may be a useful heuristic (p. 47) but there are many occasions when it could be totally unnecessary.

In general, the text is somewhat light on tools, notations, and association methodologies currently being developed and implemented by software engineers working in applications development.

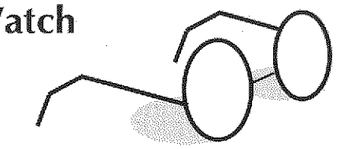
Some other minor points of concern include the feeling I have that Figure 2.9 is really more illustrative of a state-transition diagram than a data-flow diagram; that since specification of internal details of a class is independent of the abstract data type (ADT) specification, there may well be more than one implementation for each design module or ADT (p. 27); that entity-relationship diagrams (which deal *only* with data) are not, as the authors state, the most important tool with which to begin object modelling—I believe they are likely to provide some good initial ideas only; an over-early specification of *data* structure—data should be considered *per se* late in the lifecycle.

It is always easy to find points of disagreement or criticism; less easy to offer positive feedback. "Jobs well done" are too often taken for granted, especially when the job is writing a book. This book is a "job extremely well done." I congratulate the authors. ≡≡

References

1. Henderson-Sellers, B. *A Book of Object-Oriented Knowledge*, Prentice Hall, Englewood Cliffs, NJ, 1992.
2. Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

Book Watch



The "Book Watch" column does not contain book reviews. These listings are abstacted from press releases provided by the publishers, and no endorsement is implied. Please send announcements to the Editor: Robert Shelton, 1850 Union St., Stc. 1548, San Fransisco, CA 94123; fax (415) 928-3036

John Wiley & Sons, Inc. 605 Third Ave., New York, NY 10158-0012, 212.850.6497.

John Wiley & Sons announced publication of Intelligent Offices: Object-Oriented Multi-Media Information Management in Client/Server Architectures by Setrag Khoshafian, A. Brad Baker, Razmik Abnous, and Kevin Shepherd. Intelligent Offices is a comprehensive guide that shows the user how to create an environment that combines existing knowledge of digital imaging, database management, data storage, and networking for maximum information accessibility. The book provides an object-oriented model and uses readily available, off-the-shelf hardware and software in practical, real-world examples of state-of-the-art integration.

John Wiley & Sons also announced publication of Object-Oriented Information Systems: Planning and Implementation by Davis Taylor. O-O Information systems is a practical guide that cuts through the usual techno-jargon, enabling managers and executives to make informed decisions for successful object-oriented information systems installlation and development. The book explains how to purchase, build, and maintain flexible, powerful, and competitive object-oriented technologies and includes a helpful "fast track" feature which summarizes the contents of every page in the outer margins.

Methods ≡≡

From data modeling to object modeling

by Robert G. Brown

Data modeling has at its heart a very simple idea: all data is data about something. That is, about some thing.

To understand the data, understand the thing. To do that, adopt a modeling perspective. Select what is relevant and build a model of things as they exist in the world.

What is relevant in a data model is the *meaning* of the data about things of concern to the enterprise. The name not withstanding, a data model is not a model of data *per se* (as, for example, a diagram of record layouts is). To emphasize their concern with meaning, data models are often called semantic data models, information models, or conceptual models. In this article, we use the shorter term data model. A data model is used to represent and reason about the meanings of data aspects of things of concern to the enterprise.

Developing a database by modeling evolved as an alternative to analysis and design. Development by modeling consists of two phases. First, a data model is built by looking out and modeling the world. Second, the data model is transformed into a database design by applying a selection of more or less standard transforms. The models provide a concrete way to separate issues of meaning from issues of implementation. A single model can support multiple or alternate implementations for shared use by multiple applications.

Developing an object system by modeling is similarly an alternative to analysis and design. But the state-of-practice data modeling styles are too limited to support object modeling. This article looks at some of the changes and additions that are needed to support object modeling.

The description of data modeling is taken from (ref.1).

Assumptions of Data Modeling

The small banking example shown in Figure 1 uses a data modeling style known as IDEF1X.[2]

IDEF1X differs from the other data modeling styles in many ways, but not in its fundamental assumptions. All state-of-practice data modeling styles share these assumptions:

1. An entity is a person, place, or thing about which data is needed.

In the example, the entities are customers, accounts, credit card accounts, and checking accounts.

2. Data is passive, organized around entities, and shared across applications.

Anthropomorphically, a data attribute such as age represents knowledge by memory. Each customer knows its age because it has been told its age by some process.

There is no free-floating data—all data is data about an entity. A modeling discipline (normalization) requires treating data about an entity as an attribute of that entity and no other entity. For example, customer name cannot be made an attribute of account, even if statement processing would like it there.

The intent of a data model is to include the data needed by multiple systems such as check posting and credit card posting. The data is organized in a "neutral" way around the inherently shared real-world entities. Each application process is free to read and write the data as needed.

3. Process is active, organized around function, and unique to an application.

It is the separately defined processes that do all the computation and set all the data values.

Separate and distinct process models are produced usually by some variant of structured analysis. The processes are defined based on functional decomposition of larger processes. A process is not tied to an entity; it is free to access any entity necessary.

Robert G. Brown is the author of the IDEF1X data modeling technique. He is currently developing and teaching object modeling techniques. He can be reached at The Database Design Group, Inc., 441 Via Lido Nord, Newport Beach, CA 92663, 714.675.3298.

Processes such as posting checks and posting credit card transactions are naturally different at the top level. Starting out different, the decompositions tend to give different subprocesses, even if they might be potentially common. For example, posting a check needing overdraft protection involves a charge against a credit card account—as does credit card posting. But within check posting, the charge to the credit card is seen as a part of an overall process—a process that is sequential on checking, random on credit card, and involves a transfer of the charged amount to the checking account. None of these context conditions apply for credit card posting, so the charge processes are unlikely to be the same. There is no discipline analogous to normalization—nothing that forces a function to be defined one place and shared.

4. Rules are incorporated by designing processes that conform to them.

Certain very simple rules, such as uniqueness and referential integrity constraints, can be declared in a data model. The bank assigns customer numbers with the intent that they uniquely identify customers; that uniqueness rule is declared in Figure 1 as a uniqueness constraint on customer-no. But the rule that a checking account can be protected by a credit card account only if the two have the same owner cannot be declared. Such domain-specific, general rules can be incorporated only by embedding them within processes.

5. An information system is a set of applications acting on shared data to do information processing for the business.

In the example, check posting and credit card posting both act on the shared data in the data model.

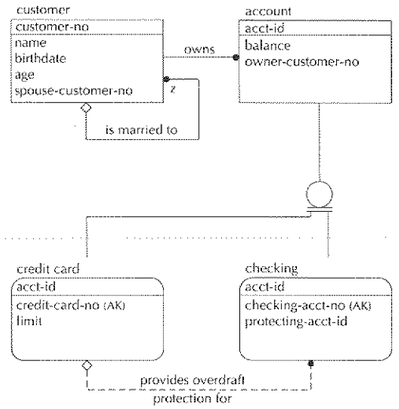


Figure 1. Data model.

ASSUMPTIONS OF AN OBJECT ENVIRONMENT

The small banking example is restated in Figure 2 as an object model. It is a model of structure—the classes, properties, and relationships. The modeling style used is described in (ref. 3).

The five assumptions of data modeling have five corresponding assumptions in an object environment.

1. An object is a person, place, or thing whose knowledge or actions are relevant.

The objects are customers, accounts, credit card accounts, and checking accounts. The knowledge needed is expanded to include derived knowledge such as whether a given amount of credit is available. Actions are added to post a check and charge a credit card.

There are two changes involved here. The obvious change is from an entity with just a data aspect to an object with an action aspect. The subtle shift in perspective is from an entity about which *we* keep data to an object that *itself* knows something we care about. We don't keep the information; we have to ask the object for its knowledge.

2. Knowledge is organized around objects and available to other objects only if they ask.

The notion of shared *data* goes away. First, it is expanded to shared knowledge, whether realized by stored data or derivation. Second, data is no longer freely available to be read or written by a process; all requests for knowledge are in the form of a message to the object. The only way to find out the age of a customer is to send a message to that customer.

Only the object knows whether the knowledge is based on stored data or derivation. A customer may know its age by memory or by subtracting its birthdate from the current date. If the knowledge is derived, the process that does the derivation is "owned" by the object and not visible to anyone else.

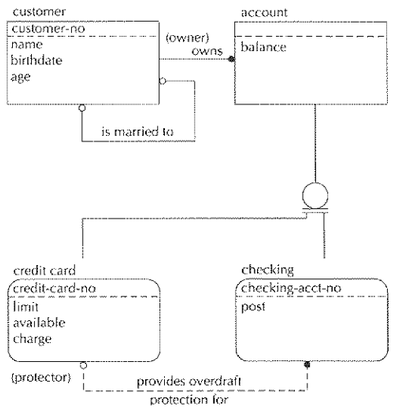


Figure 2. Object model—structure.

example implementations. Chapter 7 gives a quick introduction to C++ syntax and usage.

Chapters 9–11 contain the real conceptual focus of the book. They are unique amongst current O-O volumes and, at the same time, less strong than the earlier chapters. This does not reveal any deficiencies on the part of the authors since they are involved in the most cutting-edge research in these areas. Rather, I believe it reflects the general deficiency of work in these areas. If nothing else, I hope these chapters highlight the urgent need for a significant increase in research productivity in the areas of testing, designing for reuse, and maintenance. The authors point out (p. 51) that "Our contention is that better abstractions will be developed due to our experience with the broader domain" (viz. also considering maintenance and reuse) and that "One important goal for software design in today's environment is to produce components that can be reused. This has certainly been one of the central themes of this text. Reuse does not happen by accident."

Chapter 9 is concerned with the often neglected area of testing. It includes discussions of reuse and library management, software quality using assertions, and error handling and presents strategies for class and hierarchy testing. This has not, to the best of my knowledge, been addressed seriously in any other competitive O-O text and is certainly one of the (several) highlights and commending features of McGregor and Sykes' book.

Chapter 10 considers how to design in reuse, including a discussion of genericity, although the connection to "container" classes is not stressed nor is the very useful notion of constrained genericity (as found in the language Eiffel) mentioned.

Chapter 11 explores reusability further, providing a useful analysis of extant libraries and excellent advice for assessing the cost-effectiveness of library purchases. Indeed, it has often been said that using an O-O language requires not merely the learning of the syntax, but also gaining an understanding of the structure and features offered within the libraries that are either supplied with, or can be obtained to support, the user's chosen OOPL.

The final chapters round off the book nicely. Chapter 12 presents an example development for the software to play the game of Tic-Tac-Toe ("Noughts and Crosses"). The application of domain analysis before applications analysis I found very insightful. It places the power of reuse into a wider context than most authors, who follow through a class lifecycle and then generalize rather than, as here, permitting the *domain*, not the application, to influence the analyst's mindset. These ideas deserve further study and application.

Chapter 13 contains a disappointingly outdated view of tools (e.g., no reference to ROSE, ObjectMaker, OMTool, or OOATool) but includes some very concrete and sensible ideas for the "tools of the future." Persistency, so seldom addressed outside of specialized database books, is a very welcome contribution (Ch. 14) and the final chapter (15) is a well-reasoned and germane reflection on much-needed future work. There is perhaps some shallowness in the management problems, which are, in reality, probably greater than indicated here. The migration paths are not really discussed nor are the necessary project management tools for commercial adoption of object technology. Sadly, there is no mention of object-oriented metrics or the appropriateness (or otherwise) of structured metrics.

> ***I hope these chapters highlight the urgent need for a significant increase in research productivity in the areas of testing, designing for reuse, and maintenance.***

The book's 15 chapters progress well, adopting a spiral approach to exposition such as I also find very useful in my own teaching and writings (e.g., ref.1). However, in McGregor and Sykes' case this has unfortunately led to some internal contradictions at the detailed level between separate iterations around the same technical area. For example, literature has often been confused regarding the distinction between O-O analysis and O-O design. McGregor and Sykes give an extremely lucid description of this interface on page 47; yet, in other places (e.g., pp. 53 and 264) analysis appears to be subsumed as part of design. Further examples of internal inconsistencies: Figure 6.3c seems to violate specification inheritance recommended previously in Chapter 5. At the beginning of Chapter 4 (Analysis and High-Level Design) it is suggested (p. 59) that relationships only exist in design and not in analysis—yet this is contradicted on page 60 where the role of relationships in analysis is correctly identified. Similarly, the use of the term "O-O design" in Table 4.1 would be better as "O-O development" since relationships do not pertain solely to design but also to analysis.

There are some other small areas in which I find myself in mild disagreement with the authors (or rather with their explanation here). For example, I disagree with the statement that design of the application depends upon the existence of the appropriate classes. Application analysis is surely likely to discover new can-

BOOK REVIEW

OBJECT-ORIENTED SOFTWARE DEVELOPMENT: ENGINEERING SOFTWARE FOR REUSE

by J.D. McGregor and D.A. Sykes

reviewed by Dr. B. Henderson-Sellers

Van Nostrand Reinhold, New York ISBN 0-442-00157-6, 352pp, $42.95. Available June 1992.

Of the increasing number of O-O books being published that focus on software engineering and systems development rather than language and language development, this is certainly one of the best.

As you can tell from the title, its primary slant is that of *software engineering* for *reuse*. This is the book's main contribution to O-O literature and its main claim to uniqueness in the current crop of books. There is also a very strong exhortation in favour of specification rather than implementation inheritance: a sentiment with which I wholeheartedly concur.

The book was developed from a three-day professional development course and thus has already (presumably) benefitted from exposure to a significant number of professional software engineers. The coverage is cross-lifecycle, although naturally some areas are stronger than others. Domain analysis is clearly differentiated from applications analysis. Indeed, as the authors state (p. 337): "It [domain modelling] is a central concept for our methodology." Various lifecycle models are discussed (some only briefly) and some synergism of these views is encapsulated in the authors' own lifecycle description.

The running examples are generally good. It is pleasing to follow a particular problem through in one domain and see a solution being built up; the example presented is in a familiar domain to most readers: a graphical drawing package. The chapters are all well-balanced and generally comprehensive. The style is unbiased and the text very enjoyable to read. The content is generally (see below) bang up-to-date and includes overviews of some of the authors' own work to be published later this year in fuller detail. The authors' depth of understanding spans information systems and computer science with a hint of some of the questions currently being asked by project managers and chief executives. The slant is certainly technical rather than managerial; yet should still appeal to a wide range of software engineers concerned with learning about object orientation and, specifically, its impact on reuse strategies within their organization.

The book commences with a very balanced description of paradigms, although, surprisingly, no mention is made of Kuhn's widely accepted definition: "a large-scale and generalized model that provides a viewpoint from which the real world may be investigated." Indeed, the authors' description of the process-oriented paradigm as an umbrella above the object-oriented paradigm had, by pages 11–12, persuaded me that this was the way to go. I fear I was somewhat misled, however, as the rest of the book extols the virtues of the object-oriented paradigm!

Chapter 2 is an excellent description of basic concepts. I would quibble with the statement on page 16 that *object* is the basic concept/component: it is actually *class*. This is, however, rectified on page 19: one example of a minor, yet niggling inconsistency.

Chapters 3–6 progress through the development process. Chapter 3 describes various methodological approaches including the waterfall, spiral, fountain, and fractal models and then focuses on the authors' synergistic description. The emphasis on the class lifecycle is also very welcome. Perhaps the dangers of horizontal evolution (p. 46) cf. Open–Closed Principle/versioning should have been stressed more. Analysis and high-level design are covered in Chapter 4 as are lifecycle "activities/phases," which are sometimes *very clearly* defined and at other times a little confused. Chapter 5 focuses on lower-level considerations of relationships, messaging, and class-level details such as friends. These detailed considerations are expanded on in Chapter 6, Class Design, which contains some excellent class design heuristics. At this level, we move naturally into languages and coding considerations. There follows a quick and useful review of various languages focusing on one (C++) for the

Brian Henderson-Sellers is with the School of Information Systems, University of New South Wales, Australia. He can be reached at brianhs@usage.csd.unsw.oz.au.

3. Process is active, organized around objects, and available to other objects only if they ask.

The phrase "organized around objects" means a fundamental and radical change of perspective for process analysis. Just as all data is data about an entity, so every process is a process of an object. Posting a check to a checking account is a process of the checking account. Making a charge against a credit card account is a process of the credit card account. A process of an object can be regarded as an active property of the object.

In data modeling, the discipline of normalization dictates that a fact about an entity be expressed as a (passive) property of that entity and no other. The same discipline applied to object modeling dictates that a process of an object be expressed as an (active) property of that object and no other. At the most fundamental level, it is this discipline that leads to the increased reusability of processes in an object system. It tends to drive processes to be defined the same way regardless of the application context. In our example, the charge property of a credit card account will be usable by credit card posting and by check posting for overdraft protection.

All processing is done by the active properties of objects. An active property of an object acts by requesting the knowledge and active properties of other objects. In doing so, the objects change state and enter into patterns of collaboration. These dynamics are also modeled. Figure 3 is an example of a dynamics model. It shows some of the collaboration of a checking account and a credit card account for overdraft protection.

The initial message to post an amount, `A`, is sent to an instance of checking. That instance sends a message to itself for its balance, `B`. If overdraft protection is needed, it sends a message to itself for the identity of its protector, `C` (a credit card instance), and then sends a message to the protector to charge `A-B`. The model shows object instances and messages between object instances. The model is far from complete, but a complete model can be developed in the same way.

The tight integration of structure and dynamics models for objects stands in stark contrast to the separation of data models and process models.

4. Rules are incorporated by designing active properties that conform to them.

In our example, the common owner rule is enforced by defining the active properties that maintain the owns and provides overdraft protection for relationships so that they conform to the rule. This is a description of the current state-of-practice object environments. It is not a desirable end state. It would be better if rules could be declared directly without having to be encoded within methods. Viewed anthropomorphically, such rules would invest objects with a two-fold sense of duty. First, they would not break the rules. Second, they would act to carry out the rules without having to be told to by an explicit message.

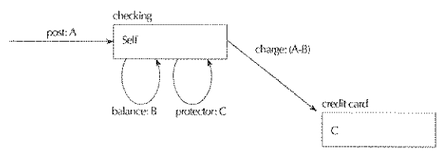


Figure 3. Object model—dynamics.

> ***The phrase "organized around objects" means a fundamental and radical change of perspective for process analysis.***

5. An information system consists of collaborating objects, each responding to requests.

The information system increasingly resembles an executing model of the enterprise. In our example, there are no separate, distinct check posting and credit card posting applications. Each arises out of collaborations of cooperating objects.

NEED DATA MODELING CHANGE?

There are many styles of data modeling. By far, the most widely used are variants of entity-relationship modeling.[4] It is toward this broad and admittedly ill-defined group of state-of-practice modeling styles that these remarks are directed.

Two naive approaches will be mentioned just to get them out of the way. The first is to say there are data objects and process objects. *Voila!* We've had objects all along. This is a just response to the "everything is an object" litany, but has little else to recommend it.

Naive approach number two is to combine the entities of data modeling with the functions of structured analysis to give objects. Trying it is sufficient to disprove it. Applications and their decomposition into subfunctions (and sub-subfunctions, etc.) are orthogonal to the shared entities of the enterprise. The functions are free-standing—they are not properties of anything. Process modeling needs a radical change from organizing around function to organizing around the things of the enterprise. The change is so dramatic it's hard to miss.

The changes needed for data modeling are more subtle. The fundamental principle—organizing around the

things of the enterprise—is already present in data modeling. Reducing the principle to practice has lead to refinements and elaborations. Some of the elaborations are appropriate for objects, some are not. It is this on-again, off-again appropriateness of data modeling that is so frustrating. It leads data modelers to think they understand objects when they don't. It leads object programmers to dismiss data modeling as irrelevant when they shouldn't.

In practice, a data model consists of diagrams and glossaries produced with the aid of CASE tools according to a methodology. To understand the changes needed requires looking beyond the practice to the underlying semantic ideas of data modeling.

Data modeling ideas

The state-of-practice data modeling styles share a set of semantic modeling ideas. These ideas are felt to track well with the way people think and therefore provide a natural way to model the semantics of the enterprise:

- The world is made up of things. A thing has a boundary, identity, and it is distinct from all other things. Joe, Ingrid, 17, and a checking account are all things.
- Similar things are grouped into classes. The classification chosen for a thing depends on one's purpose. To his wife, Joe is a husband, to his boss he is an employee, to the bank in our example he is a customer. A thing may be in multiple classes and it may be reclassified. Every thing is an instance of at least one class. For this reason, thing and instance are sometimes used interchangeably.

    1. Some classes, such as account, have a time-varying set of instances. These are called entity sets. The instances vary over time in two senses. As accounts are opened and closed, instances are created and deleted. While an account is open, its balance changes.
    2. Other classes, such as integers, have a fixed set of instances. These are variously called domains or value sets. The integer 17 cannot be deleted or changed in any way.

- Things are related to one another, that is, there are connections among things. The fact that Ingrid is 17 is a connection of two things, Ingrid and 17. The fact that Joe and Ingrid are married is also a connection of two things.
- Similar connections among things are grouped into class connections among the classes of the things.

    The class connection called age is between the class of customer and the class of integers. The class connection called is married to is between the class of customers and the class of customers.

    1. A class connection among entity sets, such as is married to, is called a *relationship*.
    2. A class connection between an entity set and a value set, such as age, is called an *attribute*.

- Class connections can be constrained in certain special ways, such as uniqueness and cardinality. The is married to class connection is constrained such that no customer can be a party to more than one marriage. Age is constrained such that every customer has exactly one age. Customer-no is constrained so that no two customers have the same customer-no.
- Whatever is connected to a given thing can be considered a property of the thing. In this way, attributes are thought of as properties of entities. 17 can be thought of as a property of Ingrid, and age can be thought of as a property of customer. Similarly, the provides overdraft protection for class connection can be thought of as a protector property of checking. Protector is the name of the role played by a credit card in a connection. The value of the protector property of an instance of checking is the identity of the connected credit card.

Data modeling has attempted (with mixed success) to incorporate the kinds of abstraction that people use every day to manage complexity. The three most prominent are generalization, classification, and aggregation:

- Abstraction by generalization considers every thing in one class to also be in another, more general class. Every checking account is also an account.
- Abstraction by classification considers a class to be a thing. Joe and Ingrid write checks for specific, physical items from the supermarket. They mentally classify all such items as groceries. When they set a budget for groceries, they are treating the class groceries as a thing—an instance of the class expense.
- Abstraction by whole–part aggregation considers a thing (the whole) to be made of other things (the parts). Joe and Ingrid's household (the whole) is made up of Joe, Ingrid, a TV, and a VCR (the parts).
- Abstraction by reification aggregation considers a connection to be a thing. Joe and Ingrid being married, a connection, can be considered a thing—an instance of the class marriage. Asking where the marriage took place exemplifies this.

There is nothing in this set of semantic modeling ideas that is directly at odds with object orientation. But

with an analogy to an area in which we have already succeeded in moving substantial human endeavor from craft focused to process focused. What is the full impact of this shift on software development?

MERLYN: A lot of things fall out of the manufacturing paradigm. Simply saying this has enormous implications: the whole notion of the software chip, application templates, and application models. Today we see companies buying entity or process models and customizing the models to fit their business. It will be far easier to do this within the manufacturing paradigm brought by objects. The granularity of the model will be small enough that we will be plugging in or redesigning a component at a time instead of entire systems. In software development, IS will begin to experience all the issues confronting any manufacturing organization: how to manage inventory and leverage it; how to address new problems with existing processes and supply; how to make the best use of existing components, subassemblies, and raw materials in stock; how to optimize stock on hand and set reorder points and quantities. Once you have the manufacturing paradigm in mind for software, we can leverage knowledge and experience already in place in our manufacturing organizations. It took eighty years to get to CIM from the introduction of manufacturing. Will it take us that long to get to CIM for software?

HOTLINE: The make-or-break issue, then, becomes the ability of IS to change?

MERLYN: Sure. The manufacturing paradigm takes software development out of the priesthood and moves it into the business domain. It's been happening for a number of years, but most IS people have tried to ignore it. With 10 million people using Lotus today, they're using stuff that would have taken a really hot APL programmer and an enormous mainframe just a few years ago. IS cannot ignore this. ≡≡

Reference

1. CIO MAGAZINE 4(11), 1991.

...continued from page 15

sure than lines of code: readability. Although for small computations, such as this, the savings in lines of code might be small, with savings of at most 1/2–2/3, the readability of the C++ code is much greater. This is especially important during the maintenance and enhancement phases of development. More complicated tasks encapsulated by zApp include Print Job, which coordinates sending the text or graphical contents of an object to a print spooler and includes automated support for printing techniques such as banding; and a Forms package, which provide a high-level interface to data collection and validation.

A properly constructed application framework can also provide portability among various GUI environments because it acts as a layer between the application program and the underlying system. We have had programmers with no knowledge of any of the native environments create applications, using zApp, that run on all of them. In selecting a portable application framework, it's important to evaluate the platforms spanned and facilities provided. Just as an application can be limited by choosing the wrong language, it is important to make sure that the framework provides accessibility to, and support for, the features of a GUI you need.

Frameworks have many other benefits, as well. Unlike conventional languages, the object-oriented languages with which frameworks are built allow the creation of new data types so that a properly designed application framework will allow easy integration of components that encapsulate and provide support for domain-specific objects and subsystems. The important concept is that these additions are as valid as the components provided with the library. This allows the developer or organization to create an environment that provides the facilities they require for their unique applications and enforces organizational standards. To that end, when we created zApp we wanted to address the issue of class documentation: how it was created and how it could be extended by the user. By embedding the class reference in the code and creating tools to extract and format the documentation automatically, we created a method that allowed both the creators and, eventually, the users of zApp to add fully documented classes to the framework that are as well integrated as those provided in the base release. It is this ability of a framework to grow and encapsulate new facilities that makes it a dynamic tool able to simplify both current and future development issues.

Over the course of our experience with zApp, we discovered something very interesting. As satisfying as creating the product was, we received greater statisfaction using it to create applications. We found that using it was an evolutionary rather than revolutionary step. It was the benefits gained from using an application framework that were revolutionary. ≡≡

MERLYN: Yes. Part of the good news in the move to workstations and PCs is that one can experiment with new technology very inexpensively. Lotus Notes, for example, can be explored on a small scale—very different from the mainframe environment. More of a natural selection process can take place. If something works, you buy more of it. You can get there gradually.

> ***If we're not good at running the old systems, how good will we be at running the innovated systems?***

HOTLINE: If the learning organization is a flexible organization, presumably it can better absorb new technology and new perspectives on technology application, such as that presented by object technology?

MERLYN: The learning organization is fundamental to being able to absorb a new technology. Historically, IS has been so unlearning. We have been the keepers of the great mystic art. Since the commercialization of the PC, the genie's out of the bottle. Now we have to step forward and catch up, and it may already be too late. In the 1985-1986 time frame, we crossed the point of having more installed MIPS on the desktop than in computer rooms. The trend away from traditional IS will continue to accelerate, driven by ever more powerful desktop systems.

HOTLINE: Let's return to business reengineering. Given our discussion of continuous improvement, are you suggesting that it has no real value?

MERLYN: Business reengineering is a wonderful idea, but it's a little bit over hyped. I'm all for it. Good analysts have tried to look at things this way in the past. Big innovations tend to cross traditional boundaries, and that is a both a strength and a challenge in successful business reengineering. I think the danger is when it is approached purely as a quick fix without strong foundations. If we're not good at running the old systems, how good will we be at running the innovated systems? TQM is the foundation for being able to innovate effectively on this scale and is used to "tune" innovations once they are installed.

HOTLINE: What do you see IS doing to build the foundation?

MERLYN: One source's data[1] suggests a high awareness of TQM among CIOs, and low idea of what to do about it. Fifty-six percent of CIOs surveyed believe that IT can play an *extremely important* role in achieving total quality; another thirty-nine percent believe the role of information technology is *important*. Yet when asked what their organizations are doing about TQM, only nine percent say they are assisting in the development of a corporate total quality plan. Thirteen percent responded that they were incorporating the principles of TQM into IS activities. The largest response, twenty-five percent, was that the role of IS was to measure quality performance. The more I work with this, the more I think the numbers are right—there are many CIOs who really don't understand it well. It seems that CIOs are more comfortable measuring other people than applying TQM to their own organizations, measuring themselves, and actively participating in TQM with the rest of the business. IS must reassess their role. The typical IS organization doesn't have the vision, doesn't think like a group of business people, and doesn't understand the business they are chartered to support. The exceptions tend to stand out.

HOTLINE: What is the impact of object technology on TQM—say, for the thirteen percent of CIOs who realize that TQM is not just a spectator sport, and must be applied to IS activities?

MERLYN: That's the good news! One of the arguments against TQM for software development is that software development has traditionally been viewed as a work of art. Object orientation is at long last the answer I can give to people who say that software development is not like manufacturing. It is. It should be. It can be. Incredible rigor and discipline has helped very committed people get there with traditional technology. Objects make it easier by encapsulation. The fundamental application building blocks are less complex. Objects dramatically simplify the application of TQM to software development by moving us closer to the manufacturing paradigm. The bad news comes back to change. IS is so resistant to change, yet they need object technology the most. It is such a fundamentally different way of thinking about systems analysis, design, and programming that there is a long learning curve, and IS has proven itself resistant to anything with a long learning curve.

HOTLINE: Brad Cox has been pushing the manufacturing paradigm for several years. Your observation reinforces that he has captured the critical transformation

data modeling has elaborated or interpreted some of these ideas in ways that are at odds with an object orientation. Examples include identity and relationships. Important new ideas need to be added, such as active properties and object abstraction.

We look next at examples of the changes and additions needed.

IDENTITY

The modeling concept of identity is the notion of uniqueness plus continuity and sameness in the face of change.[5] The identity of a thing is unique and unchanging. Joe may quit his job, get a divorce, and change his name and sex, but she is still the same customer. A modeling style supports the concept of identity if the identity of a thing is independent of its classification and independent of its property values. The state-of-practice data models do not support identity. Two entities agreeing on all property values are considered to be the same entity. This is at variance with the object model.

The lack of identity support has led to considerable confusion over uniqueness, key, and identity in data modeling. A uniqueness constraint on property values should be stated if it corresponds to a constraint in the business or is inherent in the sense of the class. The uniqueness constraints on customer-no, credit-card-no, and checking-acct-no in Figure 1 all illustrate this. A uniqueness constraint is not the same as identity. By definition, the identity of a thing never changes. But property values can change. If identity is equated with uniqueness, changing customer-no for a customer is going to be confusing. In a database, keys are used to identify records, and in a relational database a key is unique. So what starts out as a business constraint may end up being used as a key. But that does not mean that a uniqueness constraint is the same as a key. Treating uniqueness constraints as a database design issue is evidence of this confusion.

Confusion is also evident in the coupling of uniqueness, keys, and generalization. In generalization, a single thing, with a certain identity, is classified as belonging to both the subclass and the superclass. For example, a particular checking account is a checking account and an account. It is the identity of the checking account that must be the same in both classes. There is no inherent requirement that the uniqueness constraints be the same, nor the keys. But without identity support, one or the other is equated with identity and that leads to the spurious requirement that the uniqueness constraints or keys be the same. In Figure 1, the data model shows account, credit card and checking all having the same uniqueness constraint, acct-id. The need for the artificial attribute, acct-id, and its equation with identity both stem from the lack of identity support.

The source of the confusion is twofold. The state-of-practice data models do not have a strong notion of identity, and neither do relational databases. A data modeling work-around sometimes used is to mentally separate uniqueness from identity, state the uniqueness constraints, equate key with identity, and use non-updatable surrogate keys in the database. This is what was done in the data model in Figure 1. What is really needed is the modeling notion of identity, as illustrated in the object model in Figure 2.

Intrinsic identity is part of the object model. The data modeling notion of identity needs to be strengthened to that of the object model. Doing so also solves longstanding problems in data modeling. Identity support can be summarized by adding the following idea to the set of semantic modeling ideas given earlier:

- Every thing has an intrinsic identity, independent of its classification or connections.

RELATIONSHIPS

One of the lessons learned as logical database design evolved to data modeling was that the forming and breaking of relationships involves significant business semantics. The result was to elevate relationship from a simple reference to a distinct modeling construct. But the semantics of the relationship construct in data modeling do not properly account for strong identity or inheritance.

The models in Figures 1 and 2 show a relationship between customer and account. It means that an account is owned by one customer and that a customer owns zero, one, or many accounts. Suppose that Ingrid owns only one account, a checking account. If the customer instance with name Ingrid sends a message to the account that it owns, will the message go to (an instance of) account or checking? An object modeler or programmer will find the question bizarre, but answer checking. The data modeler will probably answer account—because the owns relationship is between customer and account.

What is going on here is that the object programmer is reasoning in terms of object ids, types, and substitutability and arriving at one answer. The data modeler is reasoning in terms of foreign keys (or just reading the diagram in a certain way) and arriving at a different answer.

Because data modeling has such weak notions of identity and inheritance, relationships tend to be seen as being between instances in particular classes. This is at least arguably incorrect for data models, and is certainly not appropriate for objects. In an object model, relationships should relate instances, not instances in particular classes. This is a case where many data modelers will need to rethink something that was internalized long ago.

ACTIVE PROPERTIES

Data modeling provides no way to model the actions or derived knowledge of an entity. Both are needed to address object orientation. They can be provided by extending the basic set of semantic modeling ideas in the following way. In data modeling, the existence of a connection between things is presumed to be asserted or retracted in some unspecified way. For example, an entity has a certain attribute value if that value has been previously assigned to the attribute. There is no provision for specifying the connection in terms of a computation. Adding the idea of a computed connection to the set of semantic modeling ideas given earlier allows the model to incorporate actions and derived knowledge:

- A connection may be asserted or computed.

A computed connection viewed as a property is called an active property. An active property provides derived knowledge by computing the property (output) values. In Figure 2, the age connection between customer and integer is considered the age property of customer. As an active property, age can be computed based on the current date and the birthdate.

An active property takes action by computing based on the property (input) values. The post active property (a connection between a checking account and money in our simplified example) can effect the check posting by decrementing the balance of the checking account.

The computation can be specified in a variety of ways such as state machines, a procedural language, or a declarative language. However it is specified, an active property entails object abstraction; it is not a free-standing computation.

OBJECT ABSTRACTION

In data modeling, property values of things (entities) are presumed to be set and read by free-standing computations, outside the scope of the data model. With object orientation, there are no free-standing computations. Instead, all computation is done by things (objects). This requires extending the set of semantic modeling ideas in the following ways:

- The only computation is the computation of connections. Or, equivalently, all computation is done by active properties. The only way to compute a customer's age is by the age active property of customer.
- The only way a property value of a thing can be obtained or specified is by a request (message) to the thing. The only way the age of Ingrid can be obtained is to ask Ingrid.
- The value of a passive property of a thing can be set only by the thing.

The manner of posing a request for a property value should not depend in any way on whether the property is active or passive. The separation of *what* the properties are from *how* they are realized is an important change. It allows the modeler to think in terms of the properties of the objects without having to commit to a realization. It allows the method of realization to be changed without affecting the requestors.

SUMMARY

Data modeling rests on a basic set of ideas. Most of the ideas carry forward to object modeling. But some of the ideas have been elaborated in ways that are not appropriate to objects. Examples include identity and relationships. Such elaborations need to be undone and the basic ideas brought forward in a new way. Significant new ideas must be added to the basic set. Examples include active properties and object abstraction. If the changes and additions are made, the result is object modeling. Object modeling developed in this way subsumes data modeling.

A corollary is that with (and only with) an understanding of the changes and additions, data modelers can become object modelers.

A modeling style consists of concepts, notation, and formalism. Models are motivated by goals and enabled by tools and methodology. This article has considered a few of the changes needed in the concepts of data modeling. Changes to goals, tools, and methodology are also needed.

Conversely, data modeling can contribute certain ideas not generally present in the programming heritage of object orientation. Examples include treating relationships, constraints, and rules as modeling constructs; declarative models; and the need for a consistent family of models spanning the perspectives of the owners, designers, and builders of information systems. ≡≡

References

1. Brown, R.G. Data modeling methodologies: Contrasts in styles, HANDBOOK OF DATA MANAGEMENT, Auerbach, New York, 1992 (forthcoming).
2. Bruce, T.A. DESIGNING QUALITY DATABASES WITH IDEF1X INFORMATION MODELS, Dorset House, 1992.
3. Brown, R.G. An extended conceptual modeling language for objects and rules, PROCEEDINGS OF THE 10TH ER CONFERENCE, 1991 (invited paper).
4. Chen, P.P.S. The entity-relationship model—toward a unified view of data, ACM TRANSACTIONS ON DATABASE SYSTEMS, March 1976.
5. Khoshafian, S. and R. Abnous. OBJECT ORIENTATION, John Wiley & Sons, New York, 1990.

MERLYN: Yes. To get into business process reengineering, you should first reengineer your IS function. Even before that, you should have started the journey to a learning organization—building an entire organization, IS included, that can accommodate change. TQM is the foundation.

HOTLINE: Where are we in terms of imbedding TQM in American business culture?

MERLYN: The TQM movement started to take hold in the last two to three years. Within the last year, the reengineering movement has started to surface and is tending to eclipse the TQM movement because it more closely fits the American culture.

HOTLINE: How does TQM manifest itself in software development? Has Japan, for example, been successful migrating TQM formalizations from business operations to software development?

MERLYN: The Japanese have two very different universes of software development. There are the very large suppliers of systems software, like NEC and Fugitsu; then there is the regular IS shop. The US has the same differences, and I find that those who develop large-scale commercial systems in both countries are more disciplined. The gap in Japan, however, is far more extreme. The software factories are much more advanced there than in the US. Reuse, strong team approaches, and culturally integrated TQM are fundamental. On the other hand, IS shops in Japan are five years behind IS in the US.

HOTLINE: Perhaps this should not surprise us, as the Japanese also value information technology very differently than the Americans?

MERLYN: The US puts IT up on a pedestal. The Japanese don't rely on strong information systems to create value or differentiation the way we do. They emphasize the corporate library, but not CRT-based data. To them, what's on the CRT is not interesting. That is reflected in the Japanese implementation of TQM. They track the things that are important, using a reasonable amount of automation. They are not obsessed by automation as we often are. Japanese software factories are not too automated, either. They are not carried away by fancy CASE tools. Instead, they have a rigorous process that is understood and followed, with lots of measurement, feedback, and reuse.

HOTLINE: How do we create a learning organization? What exactly must change?

MERLYN: Many things change when compared to most of today's organizations. There is less hierarchy and more control in the workers' hands. Management must show commitment to empowerment of the work force. Management sets the vision but realizes that improvement comes from the workers. Education and training in improvement techniques and team techniques is emphasized. Workers are given autonomy in using teams to achieve company goals. Management also changes motivation and reward structures. Workers in the learning organization are not just rewarded for product or service delivery, but also for contributions to the processes that produce their product or service. Rewards for process improvement become more significant than rewards for simply following the existing process.

HOTLINE: The core change seems to be a shift from management in the Industrial Age role of overseer—the origin of the word supervisor—to management as visionaries and leaders, with the workers thinking for themselves. This puts a very different burden on management, for which many managers would seem ill prepared given the lack of training provided by today's organizations.

MERLYN: This does require a very clear vision and mission for the organization. The learning organization will tend to have a much more autonomous or distributed organization structure, emphasizing self-management in teams. Without a clear vision, things can literally disintegrate. You need a clear understanding of what the mission is so all teams don't go off in different directions. The mission is not well communicated in most US organizations. It has to be very specific and very tangible. The teams and individuals need a very clear understanding of what their role is in meeting that mission so they will know what to do—remember, they will be their own management.

HOTLINE: What is the role of technology in a learning organization?

MERLYN: There are substantial changes in the use of technology in a learning organization. Technology can be supportive of a distributed, autonomous organization. Networking makes communications easier, and the easier you make it to communicate, the more people will communicate. True work-group technology doesn't just support teams but exploits them. The computer becomes an active team member in work-group computing at its best. Improved access to information is critical—it leverages the learning organization because of better access to group knowledge.

HOTLINE: Downsizing and networking would seem to work in favor of this distributed organization.

INTERVIEW

Toward the learning organization

Vaughan Merlyn, interviewed by Robert Shelton

Organizations today are caught between the temptations of innovation and the need for a more fundamental change; continuous process improvement.

This is especially true in the US, where our basic value structure encourages innovation—jumping from one silver bullet to the next has become the standing jibe at IT divisions. In fact, the ultimate business impact of our innovations depends less on the innovations themselves than on our ability to enculturate process improvement as embodied in Total Quality Management (TQM). This is where the learning organization excels, says industry expert Vaughan Merlyn, and the learning organization will have the competitive edge in the 1990s. As object technology is an innovation that presents us with a clear opportunity to apply TQM to software development, this discussion with Mr. Merlyn should be valuable to each of us seeking to balance radical change and systematic improvement as we bring object technology into our organizations.

—Editor's Note

HOTLINE: What exactly is a learning organization?

MERLYN: The idea behind a learning organization is best expressed in the Japanese concept of *kaizen*, which is this notion of continuous improvement. Everything you do you should learn from and find a better way to do it. Kaizen translates into a very personal approach to life, a fundamental approach by which to live. This is a very personal value that translates easily into the work environment. Although he started his work in the US, Deming went over well in Japan because his ideas paralleled fundamental Japanese values that were already in place. Ironically, Total Quality Management [Ed: TQM] is being overtaken in the US by business process reengineering and process innovation—just as we are beginning to realize that there is value to continuous improvement. The press is hyping that there is more value in radical change like innovation and reengineering than in continuous improvement because the basic American value is radical, sudden jumps forward. In practice, we must master both improvement and innovation.

HOTLINE: What impact is this having on our competitiveness—both as businesses and as information systems developers?

MERLYN: I do find this direction of concern. Take, for example, some of the work we are doing at Ernst & Young. Some of our clients find innovation of process relatively easy, but they cannot implement it. First, prior to undertaking process innovation, they had not created an IS organization that was sufficiently productive—that had the capacity to implement the new systems that were being demanded. Second, they had not created an organization that could accommodate change well. In contrast, the essence of a learning organization is that it accommodates change well.

HOTLINE: Given an organization able to accommodate change, why could we not arrive at similar results through innovation as through continuous change?

MERLYN: Innovation tends to bring radical change. If you're reengineering the order fulfillment process, for example, you can come up with innovative approaches. But what you then realize is that you have to rebuild three to five of your core applications that have been built over the last twenty years. That means trying to squeeze ten to twelve years of development work into six months.

HOTLINE: So the limiting factor is the speed at which IS can help support new applications for the information-dependent business?

Robert Shelton is the editor of the HOTLINE ON OBJECT-ORIENTED TECHNOLOGY. *He can be reached at 415.928.5842.*

Vaughan Merlyn is a Partner in Ernst & Young's Center for Information Technology and Strategy in Boston, MA, where he specializes in software development automation and continuous quality improvement in Information Systems. Prior to joining Ernst & Young, he was Chairman and cofounder of CASE Research Corporation. CASE Research merged with Ernst & Young in 1991. Mr. Merlyn may be reached at 617.725.1546.

COMPONENTS AND REUSE

How frameworks enable application portability

by Mark Anders

Application frameworks are libraries of object classes that provide the building blocks of an application.

They provide a directed means of applying object-oriented techniques to application development by presenting the developer with an array of reusable components that can be used as is, or extended via inheritance mechanisms, to build even the most complex applications. A well-constructed application framework can greatly reduce code size, cut development time, and increase software reliability, maintainability, and efficiency. Yet despite the substantial benefits to be gained from using a framework, many developers are slow to begin using them even as the difficulty of creating applications continues to increase at an astonishing rate.

As to why frameworks are slow to be adopted, the reason is partly the result of inaccurate perceptions of object-oriented programming: what are objects, how they are used, and how do they fit into the way that development has been done traditionally? This perception has been fostered by the purveyors of objects, who, in their zeal to spread the gospel, have positioned objects as a revolutionary breakthrough technology. Because of the hype, many developers are skeptical and reluctant to undertake what they perceive to be a radical change in the their development process. Even among those who have made the move to using an object-oriented language, such as C++, many have yet to begin using a framework. However, code reuse is one of the promises of object-oriented programming, and construction of applications from libraries of commercial components is at the heart of reuse. Frameworks are an instance of reusable components. Yet, when confronted with a framework, developers often revert to a "not-invented-here" mentality.

One example of a framework is zApp, a C++ application framework from Inmark Development Corp. that supports Microsoft Windows, DOS, OS/2, and, in the future, will support X and Macintosh. Both during development and as we've used zApp, we found that using C++ and object-oriented techniques did require a mindset adjustment but, in regards to how we create software, it was a natural one. We found that while objects are in many ways a breakthrough technology, they are not a radical departure from how software has been developed. They are an evolutionary step in development strategy with a rich history. Frameworks are an extension of the abstraction techniques we already use in software development that facilitate the reuse concepts of object technology.

To understand why application frameworks are important, and the benefits they provide, it is important to understand some of the issues facing the applications developer and put those factors in perspective.

CURRENT ISSUES IN APPLICATION DEVELOPMENT

The current trend towards the graphical user interface (GUI) has provided users with rich computing environments; however, GUIs have dramatically upped the ante for what it takes to create competitive applications and placed a great burden on those whose job it is to create them. GUIs have many tangible benefits, such as making applications easier to use and providing standards for the way an application behaves. This allows users to more effectively learn and use multiple applications and reduces training costs. It also opens up many more possibilities for making applications work together. Interapplication protocols such as dynamic data exchange (DDE) and object linking and embedding (OLE), found in Microsoft Windows, allow suites of applications to be easily integrated for a given task. Even for simple applications, the GUI has placed great emphasis on the interface from a user *expectation* point of view,

Mark Anders is Vice President of Research and Development of Inmark Development Corp. He has been developing for Windows since 1986, and is one of the principal designers of zApp, a C++ application framework for Microsoft Windows, DOS, OS/2 and OSF Motif.

so that features such as 3-D buttons and toolbars are now considered expected features of an application and not an added bonus.

The real problem with these environments is that they are hard to program for. As many who have made the move to developing software for a GUI can attest, creating programs for such as environments can be an order of magnitude more complicated than developing for other types of interfaces. The complexity lies in a number of areas. Working with bitmapped text and graphics, menus, windows, and event-driven programming all require the programmer to rethink the way in which an interface is built. New application programming interfaces (APIs) are also appearing at an alarming rate, adding capabilities for multimedia, OLE, and pen input. What we are finding is that as soon as we come to grips with ways to program for the present GUI services new features have been added. No sooner are they added, than they are expected by users. Our GUI universe is indeed expanding!

> **Neither developers nor users want to be locked into a system that does not meet their needs.**

To better understand how an application framework can address the problems facing developers, it is first important to understand what the problems are.

GUI APIs are low level

Most services provided by a GUI environment require extensive control of all details by the programmer. For example, in an environment such as Microsoft Windows a simple application, such as the "Hello World" program from Chapter 1 of Charles Petzold's PROGRAMMING WINDOWS, is approximately 75 lines of code. To use some of Windows more advanced facilities, even more code is required. The Multiple Document Interface example from Chapter 18 of the same book, a *very* simple example of how to use the MDI, is approximately 490 lines of code. Because even the most simple tasks require numerous steps, programmers working in GUI environments often spend much more time and code dealing with the peripheral details of their application rather than with the actual task that they're trying to solve.

Programming for GUIs is difficult

Having to deal with so many low-level details leads to the fundamental problem: programming for a GUI simply requires a lot of work. Though these advanced user interfaces are appealing from a usability standpoint, they do not come without a cost. An environment such as Microsoft Windows contains hundreds of API calls. Gaining an understanding of these API calls, learning which ones are important, and what happens when you use them requires a lot of time. Furthermore, having learned how to use the API, creating and manipulating menus, windows, buttons, and scrollbars require a lot of code. Also, when you consider the footprint of an application with respects to the operating system, i.e., the amount or percentage of code that is spent making calls to the operating system, GUI applications spend much of their time in operating-system or environment-specific code.

Abundance of architectures and portability

There are currently a number of different GUI environments on the market: Microsoft Windows, OS/2 1.3 and 2.0, the Macintosh, X/Motif, OPEN LOOK, and the list goes on and on. Often an organization will have to simultaneously support a number of these different GUI platforms. In other instances, there needs to be a clear migration path from current systems to other, more advanced ones. In all instances, neither developers nor users want to be locked into a system that does not meet their needs. While most of the GUI APIs offer similar services to the developer, they provide these services in slightly different ways. Unfortunately, because so much of an application running under a GUI is dedicated to controlling the various interface components, and because the APIs usually provide services at a low level, porting between environments is very difficult. The problems created by these API differences are not limited to the issue of taking an application and moving it among environments but also of taking programming resources and moving them among different GUIs.

BY WAY OF COMPARISON

As a reference point for evaluating how application frameworks provide a solution to the aforementioned problems, it's interesting to note that these problems are really not very unique. Consider, for example, the problems that faced programmers before the advent of high-level languages such as FORTRAN, COBOL, and latter C.

Assembly language is low level

Prior to having high-level languages, all software had to be written in the language of the machine, or in a basically one-to-one mapping of assembly language. In doing so, every detail of the program had to be explicitly written by the programmer. This required that the programmer approach the problems to be solved not only with an eye towards the conceptual operation to be performed, such as adding two numbers, but also with regards to exactly how this was to be performed on the hardware level. For example, on many machines two values can be added only when contained in the CPU registers. Therefore, if the quantities to be added are contained in memory locations, they must first be moved to CPU registers and then added. Other common operations such testing logical conditions and performing a set of instructions based on the result (if...then...else) also could require many steps.

Programming in assembly language is difficult

The general problem with assembly languages therefore becomes that by forcing the programmer to handle every detail at the machine level—i.e., by requiring that not only the problem of what logical operation is to be performed but also *how* it is to be performed be solved—the task for the programmer is complicated to the point that many problems become too difficult to solve.

Abundance of architectures and portability

Another major problem of coding in assembly is that because a program was written in the language of the machine, as new machines with different architectures were created software had to be completely rewritten. While the instruction sets of many of these machines had similar concepts, such as adding two numbers, or moving data from a memory location to a register, they implemented them in different ways.

THE HIGH-LEVEL LANGUAGE SOLUTION

The solution to these problems was high-level languages, which encapsulated common operations performed in creating a program into expressions that were easier for the programmer to manage. What the creators of high-level languages realized was that by allowing the programmer to express at a higher level what a program should do in a natural way, such as "3 + 5", software development would be greatly simplified. Thus by removing the burden of dealing with how the machine performed the actions, more complex problems could be solved. It was also discovered that by eliminating the machine-dependent operations from a program—creating a level of abstraction between how the machine needed to perform a task and how the programmer expressed the task—programs could be easily moved among machines of different architectures.

APPLICATION FRAMEWORKS

In much the same way that high-level languages simplified the programming process by providing a more powerful and more concise interface to the instructions that the programmer wanted the computer to execute, the object-oriented structure of an application framework simplifies GUI application development by encapsulating common operations and interface behavior into prefabricated objects. To exemplify user interface objects provided by an application framework: zApp provides over 20 different types of windows including main application windows, push buttons, list boxes, and edit boxes. Additional classes include, but are not limited to, objects that correspond to various entities typically associated with a GUI including graphics display devices such as window, bitmap, and printer; drawing tools such as pen, brush, and font; and geometric entities such as points, rectangles, and clipping regions.

Using these objects to build applications yields measurable results. The "Hello World" example for Microsoft Windows, previously mentioned, which had required 75 lines of C code, was created with zApp in 11 lines of C++ code. The Multiple Document Interface example that took 490 lines of C code was completed in approximately 100 lines of C++ using zApp.

Objects are also provided that encapsulate common operations. Consider the following. You're creating an application that displays text in a 40 pt. font. To create that font, you must determine how many pixels there are in 40 pts. for the device on which you will display the text. The first step is to inquire from the device the ratio of pixels/inch. You then must realize that there are approximately 72 pts./inch, multiply the pixels/inch ratio by the size of the font (40), and divide by 72. With zApp, you simply create an instance of a zPrPoint object, a logical sizing and dimensioning object representing pt. units. You provide it with its size, 40, and the display object with which it is to be associated. It will perform all the calculations. This example also hints at a benefit of an application framework that is more difficult to mea-

> **It is the ability of a framework to grow and encapsulate new facilities that makes it a dynamic tool able to simplify both current and future development issues**

continued on page 19 ...

≡ 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'd like to see but don't — or whatever other feedback you'd like to send. Send comments to Robert Shelton, 1850 Union Street, Suite 1584, San Fransisco, CA 94123; Fax: (415)928-3036; 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	❑	❑	❑	❑	❑
Methods	❑	❑	❑	❑	❑
Standards	❑	❑	❑	❑	❑
CASE	❑	❑	❑	❑	❑
Book Review	❑	❑	❑	❑	❑
Product Announcements	❑	❑	❑	❑	❑
FYI	❑	❑	❑	❑	❑

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

VOLUME 3, NUMBER 9 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES JULY 1992

OOD: research or ready ?

Objects are entering the mainstream application and system-development community.

As this continues, experienced managers anxiously seek effective design methods and CASE tools to complement object-oriented languages.

Development managers have only a passing interest in research projects and research methods. To them, selecting a method means they're betting their next major project on it. As managers, most of us have been subjected to enough software that wasn't system tested or, in some cases, even unit tested prior to release. I, for one, have no desire to knowingly repeat that experience, especially not with a new design method.

With these experiences in mind, this article describes a commercial screen for evaluating object-oriented design methods and CASE tools. A conscientious effort has been made to provide the most accurate and current information about all three of the methods described, but this is a rapidly changing field. The most important message of this article is "don't be afraid to ask your own questions."

Patty Dock

A COMMERCIAL SCREEN

I recently had the opportunity to take a fresh look at the current state of object-oriented design methods. My first observation concerned the immaturity of some of the methods. It was apparent that some authors were simply unfamiliar with objects.

Back in the good old days of structured programming, the recognized wizards, who had been involved in building several successful systems with their new methods, wrote the first wave of books on design. Such a grounding in practical system building is exactly what appears to be missing in many of the design books for "the new way of programming."

As an experiment, I made a list of what I considered the absolute minimum requirements for an object-oriented design method to be used on a commercial project. Then I started reading, asking questions, and making telephone calls to both vendors and users.

My list contained the following fundamental questions:

1. Is the method described in an English-language book readily available to a team of developers?
2. Has at least one author of that book participated, full time, in an object-oriented development project?

continued on page 7 ...

IN THIS ISSUE

From the Editor

True, Business Week made "object-oriented" a board-room word with their September 30, 1991, cover exposé "Software Made Simple," but the downside risk is that our nontechnical colleagues have externally validated, preset expectations that the adoption and application of object technology is child's play. In reality, making the "thinking" transition from procedure-think to object-think is a significant challenge. Actually delivering successful designs for component reuse is a much more daunting task altogether. Shall we broach the subject of business analysis?

Saying that objects remove the impedance mismatch between analysis and design because programming objects can be more like real-world business objects ignores a more essential truth: our limited capabilities for understanding business objects leaves most practitioners of object-oriented technologies with prototyping and a rapid development cycle as their principle lines of defense against complex business problems.

At one time business partners would have expected us to turn to computer aided software engineering (CASE) for help. However, Information Week's February 17, 1992, cover exposé "The Case Against CASE" has helped to slam a door that was closing fast already. Granted, it was CASE vendors and software technologists themselves who created the CASE disappointments and disasters we have seen over the last five years. Note too that, despite this article, the concept of CASE tools is still far from dead. But we do find ourselves with a double-edge sword in hand: Business Week tells our business partners that this new technology is software made simplistic; Information Week tells the few still-employed CIOs that CASE is a lost cause. Meanwhile, we are supposed to implement the results on schedule and within budget.

Sounds like computer business as usual to me! So what's new with objects?

Attendees of Object Expo, Object World, OOPSLA, and even the more general industry shows like DB Expo and Software Development are well acquainted with the plethora of CASE tool vendors, training-seminar vendors, and even development-environment vendors offering methods to support object-oriented software development. Among these offerings, the most common approach is to extend existing methods, modeling techniques, and CASE tools to support some subset of ideas associated with object-oriented programming. More sophisticated approaches actually tackle design from the viewpoint of an object, recognizing that the "-oriented" part of this technology's name is about thinking. Among the players claiming to support object-oriented analysis, perhaps the kindest observation is that few have expressed a clear understanding of the problem space, whereas, as an industry, we're still wrestling with solution space, i.e., design.

Historically, as John Zachman has frequently observed, we computer technologists have worked our way upward. We start with programming the machine, and eventually discover design, analysis, and (voila) the enterprise! His assessment is as valid applied to object technology. Development environments are well established. A bevy of solid language implementations is available. Object database management systems are available from several vendors that can actually handle modest-sized applications. Some very good class library products are available to solve technical problems, and several companies threaten to introduce business object class library products later this year. We are now ready to discover the need for design!

It's not that design issues and methods have gone unconsidered over the last several years. Just the opposite is true. It is only in the last 6–12 months, however, has design become an unavoidable issue industry-wide. Not surprisingly, only during this same period have enough developers used object technology on projects serious enough to get themselves very visibly in trouble! Several vendors have suggested to me that the lack of "a method" could even be an impediment to their sales efforts. This is a far cry from projects only a year ago on which design methods were considered—only to be discarded as adding unnecessary complexity to the already daunting task of learning object-oriented programming. Besides, at this point we as an industry finally

continued on page 3…

hotline on OBJECT-ORIENTED *technology*

The Hotline on Object-Oriented Technology (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012, (212)274-0640. Mailed First Class. Subscription rate — one 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 Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123. Voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of *Hotline on Object-Oriented Technology, Journal of Object-Oriented Programming, Object Magazine, The X Journal, The C++ Report, The Smalltalk Report,* and *The International OOP Directory.*

The Business of O-O

…The genius of object oriented programming is that once a good idea has been realized, it can be shared. "If the language guys do something neat," says [Borland's] Neil Snyder, director of product marketing in the spreadsheet business unit, "we can say, 'Damn. We like that. We're going to steal that for Quattro Pro.'"….

Borland bends the rules, Mark Hindley, ***VAR BUSINESS****, 4/92*

…[InfoWorld's Lisa] Picarille: You said Borland has not suffered any product delays. Yet Bill Gates has been quoted as saying "If object orientation and C++ are so great, why hasn't Borland delivered yet?" That has made Wall Street analysts skeptical. [Borland's Philippe] Kahn: Three years ago we made a huge bet and started Quattro Pro and Paradox for Windows from scratch—brand new code bases, no compromises, new architectures, new development methodologies, the works. Object orientation requires a large investment of time and effort up front, but the payoff is not only better products but more efficient development in the future, and it will give us a five-year technology leadership position….Borland's products are not late—they're early and the first of a new generation….

Merger or no, Borland's Kahn plans to beat Microsoft, Lisa Picarille, ***INFOWORLD****, 4/2/792*

Distributed Environments

…"Object technology offers a second-generation model for client/server, with a clear role for a powerful client as well as a powerful server," said David Gilmour, executive vice-president of sales and marketing for Versant Object Technology, Menlo Park, Calif. By raising the power of an individual object to support transparent peer-to-peer communication via messages, the idea of client/server extends to a more robust notion of objects. Under this notion, objects could at one point make requests as clients to servers, then at other points act as server to other clients. This allows a modular distributed system that may be more responsive to change. Using objects as the unit to be distributed may allow developers to save implementation issues—such as distribution—until after the design is complete. "This is because object technology is an inherently parallel technology that naturally thrives in a distributed multiprocessing environment," said Dr. David Taylor, principal of Taylor Consulting, San Mateo, Calif.…

Objects can set the stage, Eric Aranow and Tom Kehler, ***SOFTWARE MAGAZINE****, 5/92*

Databases

…In the world of textual data, relational databases worked fine. Text gives you structure and form in the way of character strings and numbers. This is something an RDBMS can handle quite well. Unfortunately, when you start dealing with multimedia data types—where you have to deal with massive amounts of this data, many of them being object-based—an RDBMS falls flat. By contrast, object-oriented databases come out way ahead of RDBMSes when dealing with heterogeneous, complex data involved in complex relationships. More importantly, when you start getting applications designed to integrate these multimedia data types into their programs, it will be important for them to include, as a part of the applications, an object-oriented database to help them handle these new types of object based data. At first, you will see these object-based databases added to authoring products, then to presentation, drawing and desktop-publishing products. They will also become important to any word-processing and next generation on-screen document communications. Ironically, it will not be the traditional database suppliers that will help these independent software vendors use a database effectively in this multimedia-driven world. Even though they all have object-based databases in the works, unless they are able to perfect them soon and make them work harmoniously with their RDBMS programs of today, they could be left out in the cold. In the future, the database will be embedded in major applications so they can manipulate these stored images, video and sound and integrate them into the heart of the app. Whether anyone likes it or not, multimedia computing is going to revolutionize the way we use computers.

The soft view: Multimedia simply spells a new digital data type, Tim Bajarin, ***COMPUTER RESELLER NEWS****, 4/20/92*

Standards

…A great deal of important work involving object-oriented system management currently is underway by a number of standards organizations. From these organizations will come the definitions for the framework, common facilities, object transportation and definition layers. Various groups also are involved in the standardization of the object definitions themselves, such as user, system, group, printer, print job and device objects…System management standardization, especially the transportation layers, the framework and common facilities, and the object definitions, are vital to the success of distributed systems management. Without these standards, management applications and object definitions will quickly diversify, until they are as cumbersome to use as existing vertically integrated solutions. Both the industry and the standards bodies themselves know of the critical need to build generalized management applications that use a standard framework and a standard set of managed objects. The outlook, however, of both these groups can be quite different….

Unix system management: new developments, Geof Bullen, ***NCR CONNECTION****, 4/92*

Tools

…Until fairly recently, C++ was viewed by many as simply an "academic" language because of its lack of good tools. Now that tools for C++ are improving, many software development teams are ready to make the shift from C to C++. Several compiler vendors are currently offering full C++ compilers that generate object code directly from a C++ program as opposed to just translating it into C code and compiling it with a C compiler…Today's selection of C and C++ compilers for the PC ranges from highly optimized compiler-only solutions to complete integrated development packages that contain debuggers, profilers and extensive class libraries for C++. Most of these compilers cost under $1,000, making it feasible to buy several and test them out at the same time…

Product Focus: C/C++ compilers bring faster code crunching to PC platforms, Jeffrey Child, ***COMPUTER DESIGN****, 4/92*

F.Y.I.

Excerpts from leading industry publications on aspects of object technology

Creative Implementation

...If no one knows what is going on inside an object's functions, and no one can tamper with its data without authorization, then an object is highly secure. It polices its own borders, responding only to authorized messages...Since an object has boundaries, you can own it. You can reward or punish the persons who designed it. You can rent out the use of the object without telling how it works. You can see a certain appeal here to the corporate mind....

Object-oriented programming: What's the big deal?, Birrell Walsh, ***MICROTIMES****, 3/92*

...I've discovered that the single greatest challenge of tackling a new object-oriented program is keeping a vision of the program that's accomplishable. As I was writing this program, I have to admit that at times I was thinking of an interactive CD-ROM-based multi-media extravaganza. Luckily, common-sense and deadlines prevailed...

...Templates are a major C++ facility and should be explored by every programmer interested in extensibility or data structures. They have gotten short shrift in most C++ tutorials, being a little too advanced for beginner books and too basic for advanced books....

Expert's toolbox: Templates of Doom, Larry O'Brien, ***THE CHICAGO PURCHASER****, 5/92*

Analysis & Design

...In the Object-Oriented (OO) Revolution the development paradigm offered appears to be "Analysis is finding and decomposing objects, while design is finding sub-objects and decomposing them in greater detail." OO practitioners further argue that this process should be documented in pseudo-code or better yet, in an Object-Oriented Programming Language (OOPL) so that a continuum exists from analysis through design into implementation. The result is a further blurring of the boundaries and a shift in focus towards code. The problem is that the closeness to code causes confusion with code and a tendency to describe how rather than what. This leads to premature implementation before the requirements are clearly understood and the design architecture is properly analyzed. This is a charter for hacking systems into existence. Perhaps we should rethink the "Fountain" life-cycle....

Analysis versus design: is there a difference, Clifford Inwood, ***THE C++ JOURNAL****, Vol. 2/No. 1, 1992*

...[Rob Dickerson, VP and General Manager of The Database Business Unit at Borland International]: I think you've got to learn to do a class hierarchy. The first time you do your class hierarchy, you write out what looks obvious, and you fiddle with it, and you realize it's not the best one. So you redesign it, and by the time you're done, the class hierarchy you end up with was not what you initially thought. And there's a bunch of tricks to it—how to identify a meta-class, factoring, the notion of collection classes, how to design a class hierarchy, but that's the main design effort. At least, that's what I've seen our R&D guys have to get their hands around. [Jacob Stein, Chief Technologist for Servio Corp]: And there's lots of trade-offs, trade-offs between reusability, and a natural fit to the system you're modeling. They might not always be exactly the same. There may be a trade-off between designing for reuse and designing for this particular application, and you have to take that broader scope. It's said that people don't get classes right until they've been implemented about three times, which might mean that some of the interfaces will change during that course of time....

Roundtable: Experts speak on object-oriented development!, John L. Hawkins and Dian Schaffhauser, ***DATA BASED ADVISOR****, 4/92*

Strategies

...In a 220-page report on The Future of Software published last year, ButlerBloor, a European research and consulting firm, warned that most large companies are fashioning a "software time bomb" in that the more systems "they generate, the more there is to maintain." It identified object orientation as a new means of defusing the problem...To get to the object-oriented multimedia era implies that firms must first discard 20 to 30 years of IS investment, says [Computer Associates International's Dominique] Laborde. He says software companies that provide object orientation as an extension to existing databases will be the successful vendors of the 90s.

The great new database debate, Ron Glen, ***CANADIAN COMPUTER RESELLER****, 4/15/92*

...At Raytheon Missile Systems Division in Bedford, Mass., a 1984 study of 5,000 production applications found 40 percent to 60 percent of their code was subject to standardization via reusable structures. Following an encouraging pilot test, Raytheon's use of standard modules and structures in a COBOL environment yielded an average of 60 percent reusability. On average, this reduced development effort by 50 percent and work effort by 70 percent during maintenance. These results were achieved, furthermore, with a language that did not couple reuse of code directly with the semantics of the language, as would an OOP tool. These figures can be viewed as pessimistic estimates of the gains that can be achieved by reusing code with OOP, following startup efforts....

Lower life-cycle and maintenance costs make strong case for OOP, Peter Coffee, ***PC WEEK****, 5/18/92*

...continued from page 2

have enough real-world experience to be expected to make a meaningful statement about design.

As Mr. Zachman suggested, serious attention is also being turned to analysis, and a few thinkers scouting the fringe would suggest that there may be more to object-oriented modeling at the business enterprise level than meets the eye. Thus, it seems timely to focus this issue of the HOTLINE on methods and tools issues and industry trends that will substantially impact how we develop object-oriented software.

Our feature author this month, **Ms. Patty Dock** of Orgware, Inc., surveys current methods for object-oriented design and finds the field of strong contenders surprisingly limited. Starting with a very reasonable set of basic filter criteria, she winnows twenty-two on-the-market methods for object-oriented design down to three viable contenders for object developers today. While this survey does not cover every commercial method of which I am aware, it certainly covers the name-brand marketplace including those companies the principle business of which is selling methods or analysis/design tools. That only 14% of the "major" approaches to object-oriented design pass a simple sanity test gives one pause for thought!

Mr. Christopher Stone, President of the Object Management Group (OMG), discusses this industry-standards-setting organization's 12–18 month operating plan. Many view object technology (and object database management systems in particular) as central to the cost-effective and timely development of the next generation of CASE tools. The OMG, as the primary standards-setting forum for the object industry, continues to actively pursue the development of a framework in which, among other things, the integration of CASE tools and class library products from different manufacturers could become reality for the business application developer. With the Common Object Request Broker Architecture (CORBA) Version 1.1 in place, a first step has been taken toward the tooling vision expressed by Ms. Hardman in her article, this issue.

From Canada, **Mr. Richard Dué** tackles enterprise modeling from the vantage of object orientation. Mr. Dué proposes the object perspective as an alternative to some of the well-recognized deficiencies of traditional enterprise (data) modeling. While as a practicing modeler I share his conviction that object technology removes critical problems, such as the nonsemantic segregation of data-dependent process from the data on which it operates and the creation of data structures devoid of meaning and policy, experience suggests that some of the most fundamental impediments to enterprise modeling will not be resolved by modeling objects instead of entities. As Mr. Dué suggests, most organizations today do not have a strategic direction and semantic modeling inevitably ignites dormant political disputes that reflect this deficit. Furthermore, many modelers fail to understand the significance of semantics, and data is commonly confused with information. Objects, however omnipotent, will not resolve these most human of problems.

On the bright side, the simple mechanism of removing the above-mentioned process-data impedance mismatch does simplify enterprise modeling in at least two areas. First, the underlying metamodel is far simpler for an object model than for the separate processing and data models. Structured analysis couldn't mate these separate models at a methods level, much less in the CASE tool repository. The simplification of the metamodel alone could bring repository implementation within reach. Secondly, the parallel between enterprise objects and programming objects can probably constrain the meaningful relationships between business objects and implemented classes to an understandable one-to-many mapping. This would be a welcome alternative to the pre-object, many-to-many "maybe-mapping"—the "maybe," of course, because we could never be quite sure....

Back to reality from theory, though, it remains for us modelers to prove that our view of object-oriented reality modeling can truly overcome the problems that, to date, have been inherent in enterprise-wide modeling. Where top-down approaches have been criticized as inflexible, time consuming, and politically unpalatable, development-seasoned practitioners suggest that objects offer a bottom-up alternative. However, they seem inexperienced when it comes to taking on the business planning and interpersonal problems that have motivated top-down modeling approaches. This is the state-of-practice in our industry today. From the eventual blend will come our strength, and Mr. Dué proposes some criteria for making the mix work.

Also from Canada, **Ms. Sue Hardman** of COGNOS sets her sights on the future of CASE tools. Her multidimensional vision for a thinking-companion tool is, in the estimation of this editor/practitioner, right on target. Two-dimensional static models have appeal for static structures. The success of such models in architecture lead many of us to pursue two-dimensional process and data models—never mind the integration problems. Unlike buildings, systems and businesses are structurally dynamic. Businesses today must do to their systems what would be the equivalent in building construction of completely redesigning the core of a modern high-rise. Construction engineers would give the same bad news that many information systems organizations have been forced to deliver: start over! That's not economical today—ergo our interest in object technology wrappers around legacy systems and object databases as semantic integration platforms.

Furthermore, Ms. Hardman strikes on an issue of correctness—using visualization to understand both what should be happening in the business and what is happening given the way we have designed our system. This reality-check could be a viable alternative to mathematical correctness—something that, however right, is impractical for many of the normal human beings involved in building business systems.

Back from the future, one of our regular book reviewers, **Mr. Michael Fuller**, reviews the second of the pair of books from Sally Shlaer and Stephen Mellor, OBJECT LIFECYCLES: MODELING THE WORLD IN STATES. Mr. Fuller's determination parallels Ms. Dock's assessment of the Shlaer/Mellor method and CADRE tool implementation. That these opinions were arrived at completely independently of one another suggests that we should be cautious of the myriad of structured-cum-object approaches being marketed today. Consider Mr. Fuller's insights on the subject beyond the context of a review of one book—he may have given us insight into a class problem. It seems clear there remains a genre of less-than-effective approaches that have grown out of the split process/data-structured analysis world seeking self-preservation.

Of course, having seen the prematurely shrinkwrapped OMTool that GE Advanced Systems Division was headlining at Object Expo in New York, we could as well apply the same caution to the Rumbaugh method. And Rumbaugh was one of Ms. Dock's qualified 14%! Look very carefully before you buy!

Armed with our authors' sharp-minded thoughts, we are better able to evaluate our alternatives as we seek to upgrade our application-development process model and toolset with an eye for eventually doing effective enterprise-wide business analysis.

METHODS ≡ ≡

Enterprise modeling: an object approach

by Richard T. Dué

Information engineering attempts to describe an integrated view of all data necessary for the operations of an organization.

This integrated view, or enterprise model, is considered to be necessary for the planning, implementation, and control of relational database management systems. Typically, in a process that can last months or even years, the organization's goals, critical success factors, functions, resources, data, and data relationships are collected. This is accomplished through a series of interviews, joint analysis and design (JAD) or joint requirements planning (JRP) sessions, by the analysis of data dictionary usage statistics, or, in some cases, by comparing the organization under study to preexisting models of similar organizations.

Richard T. Dué is President of Thomsen Dué and Associates, Ltd. in Edmonton, Alberta, Canada and a member of the X3H7 Object Information Management Standards Committee. He can be reached at 403.439.4627.

The outputs of this process are one or more graphical or narrative models that describe the organization's strategic, tactical, and operational data and data-processing requirements. Data and data-processing functions are defined in glossaries, data dictionaries, encyclopedias, and repositories. Data and data-processing functions required by the organization are mapped against existing capability. Gaps, redundancies, and conflicts between required and existing systems are identified. Responsibility for the creation, retrieval, updating, and deletion of data is aligned with the organization chart. Plans for the development of new systems and the integration of existing systems are developed and prioritized.

The main purpose of this approach has been to assist the data-processing section of the company to plan for information technology requirements, design databases, and set systems development priorities. In some cases, however, where senior management has been committed to and thoroughly involved in the enterprise modeling process, the process of building an enterprise model has given the participants a deeper understanding of how the organization actually operates. Forced to look at the organization from a strategic perspective, many participants report that for the first time they understand how their enterprise is organized and how it performs. This data-centered enterprise-modeling process is generally performed in a top-down manner, producing a relatively static description of the organization captured in narrative, CRUD matrices (tables showing organizational responsibilities for the creation, retrieval, updating, and deletion of data), and diagrams (entity-relationship, data-flow, state-transition, network, architecture, and organization charts, etc.).

LIMITATIONS OF THE INFORMATION ENGINEERING APPROACH

Unfortunately, the information engineering approach to enterprise modeling suffers from a number of shortcomings.

It takes too long

Scarce and valuable people in the organization are forced to spend months or even years of their time trying to develop the models. The concepts and techniques of this approach to enterprise modeling are unfamiliar to senior management, users, and even data-processing personnel. Dormant political disputes over data definitions, data ownership, process definitions, etc. can erupt as the information-engineering process focuses attention on redundant, inefficient, or poorly planned areas of the enterprise, slowing down or even stopping the entire exercise.

People in the organization do not see the value of the exercise

Information-engineering enterprise models require substantial further effort and investment before the underlying databases are implemented and before individual applications can be built. Information engineering's centralized approach to planning can conflict with decen-

OO Option

Micro Focus, 2465 East Bayshore Rd., Palo Alto, CA 94303 415.856.4161

Under its Early User Program, Micro Focus is shipping the Object Oriented (OO) Option to its Micro Focus COBOL Workbench. The **OO Option** provides Workbench customers an opportunity to experiment with object-oriented languages and development environments and provides tools for the COBOL programmer including two object-oriented development environments: a Runtime Environment (RTE) and a Reusable Code Manager (RCM). Micro Focus Early User Programs are designed to bring new technology to Microfocus customers who are willing to experiment and provide feedback on products, which helps to refine interfaces and other underlying features prior to full release. The Object Oriented Developers Kit is the key component in OO Option and contains the Micro Focus OO COBOL environment. Smalltalk/V PM is also a component of the OO Option, allowing programmers to write code with Smalltalk/V PM and then use the OO Developers Kit's extensions to the Smalltalk/V class library to create communication objects. These objects permit Smalltalk and COBOL programs to work together in a client/server mode of operation, the Smalltalk and COBOL environments remaining mutually independent.

New language releases

Borland International, Inc., 1800 Green Hills Rd., P.O. Box 660001, Scotts Valley, CA 95067-0001 408.439.4825

Borland International, Inc. announced the availability of several new language releases. Taking advantage of the advanced features of Microsoft Windows 3.1. and adding features to Borland's C++ and Turbo Pascal for Windows product lines, including full Windows 3.1 and 3.0 support, color syntax highlighting, and new Windows documentation. **Borland C++ 3.1** is a complete development system. **Turbo C++ for Windows 3.1** offers a route to Windows programming for ObjectWindows level C and C++ users, including the basic tools necessary for creating Windows applications. **Turbo Pascal for Windows 1.5** now lets users take full advantage of the new features in Microsoft Windows 3.1 including OLE, common dialogs, drag & drop, and Truetype fonts. Borland has also introduced ObjectVision 2.0 for OS/2.

VOSS/Personal

Logic Arts, 75 Hemingford Rd., Cambridge CB1 3BY England +44.223.212392

Logic Arts introduced **VOSS/Personal**, a low-cost personal version of the Virtual Object Storage System for Smalltalk/V. It is available in two versions: VOSS/286 Personal and VOSS/Windows Personal for use with Smalltalk/V 286 and Smalltalk/V Windows, respectively. VOSS/Personal is fully compatible with the equivalent main product line and can read and write the same virtual object spaces, providing transparent access to persistent Smalltalk objects of any class on disk without the need for a separate DBMS programming language. It has the same transaction management of updates, the variable-size cache of virtual objects in the image, and most of the same VirtualDictionary and VirtualCollection classes for managing collections larger than the image.

PARTNERINGS & ACQUISITIONS

Symantec Corporation has acquired **MultiScope Inc**. and **The Whitewater Group Inc**. The acquisition of MultiScope is a pooling of interests in which Symantec will issue approximately 165,000 shares of its common stock for the current outstanding shares of MultiScope stock. The acquisition of The Whitewater Group is a pooling of interests in which Symantec will issue approximately 80,000 shares of its common outstanding stock for the current outstanding shares of The Whitewater Group stock. Personnel of both companies will remain in their current locations and report to Carol Clettenberg, the newly appointed Director of Development Tools at Symantec. Mark Achler will join Symantec in the position of Director of Visual Tools. Chuck Duff will become Principal Architect reporting to Mark Achler.

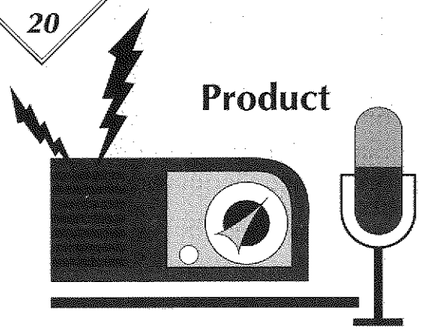
Zinc Software, Inc. announced the establishment of its first European office. Paul Leathers, former president of Zortech Ltd. and Zortech Inc. will serve as Managing Director of Zinc Software (UK) Limited, based in London. Zinc (UK) will assume responsibilities for European operations including sales, distribution, and technical support for Zinc's product line.

Object Technology International, Inc. (OTI) announced that Knowledge Systems Corporation (KSC) has been appointed as a U.S. distributor for ENVY/Developer, its object-oriented product development environment.

Servio Corporation and **Hewlett-Packard Company** announced that Servio has been named an HP Value-Added Business Partner. The companies also announced that Servio's GemStone object database and GeODE object development environment will be made available for the HP Apollo 9000 Series 700 PA-RISC-based workstation family.

Franz Inc. acquired the rights to Procyon Common Lisp for MS-Windows and other operating systems on personal computers. Franz will further develop, market, and distribute the product worldwide under the name Allegro. Procyon Common Lisp was developed by **Procyon Research Ltd.** of Cambridge, UK, a subsidiary of **Scientia Ltd.** The Common Lisp product for MS-Windows will feature CLOS (Common Lisp Object System).

Gain Technology entered into a development-assistance agreement with **IBM Corporation**, providing Gain's multimedia software for the IBM RISC System/6000. **Sun Microsystems Computer Corporation** and Gain Technology have also entered into an agreement under which Sun will build and distribute Gain-based multimedia applications on Sun workstations.



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.

C++ By Design

SIGS Conferences,
588 Broadway, Ste. 604,
New York, NY 10012
212.274.9135

Grady Booch, the developer of the Booch Method and O-O analysis and design method, and Bjarne Stroustrup, the originator of the C++ programming language, are teaming up for a four-city educational tour. **C++ by Design** is sponsored by the C++ Report and organized by SIGS Conferences. The one-day instructional seminar series is scheduled for September 1992 in New York, Chicago, Dallas, and San Jose.

InSession 3270

Avatar Corp.,
65 South St.,
Hopkinton, MA 01748
508.435.3000

Avatar Corporation is shipping **InSession 3270** for NeXT computers and the NeXT Programmer's Toolkit. InSession 3270 includes an external hardware unit and software that provides IBM 3278/3279 terminal emulation and file-transfer capabilities for NeXT computer users. It supports IBM terminal models 3278/2-5 and 3279/2A, 2B, 3A, and 3B. InSession 3270 provides direct coaxial connectivity and is attached to the SCSI port of the NeXT computer and to an IBM control unit in the host environment via coaxial cable. Applications can be developed so NeXT computer users can transparently access IBM host information without leaving the familiarity of the NeXT environment. The Toolkit includes software modules that create the link between the NeXT Interface Builder objects and Avatar's InSession 3270 software. Developers have access to C and Objective C languages in addition to NeXT Interface Builder and SpeakerListener development tools.

ODBMS 1.0 for Objectworks/ Smalltalk

VC Software Construction,
Petritorwall 28, 3300
Braunschweig, Germany
+49 531 24 24 00

VC Software Construction announced **ODBMS 1.0 for ParcPlace Systems Objectworks/Smalltalk**, an object-oriented database management system supporting most Smalltalk languages. Its storage facilities for objects can be used during the development of Smalltalk applications and stand-along database applications.

VC Software Construction also announced enhancements to other releases of ODBMS. ODBMS 1.02 Programmer's Version now offers support for Digitalk's Smalltalk/V 2.0 for Windows and for OS/2 2.0, as well as online documentation, enhanced search by using object names, and a new class hierarchy for Proxy-Objects. ODBMS 1.02 Graphical User Interface now has graphical representation of objects and object links, drag and drop, an interactive Help facility, and MDI support (Windows only). ODBMS 1.0 is now available as an educational version for Windows and includes the ODBMS Programmer's version, the ODBMS/GUI, and the Distributed Smalltalk Software Development environment (DSSDe). The education version contains most of the functionality of the commercial versions of ODBMS.

Smalltalk/SQL

Synergistic Solutions Inc.,
63 Joyner Dr.,
Lawrenceville, NJ 08648
609.586.0025

Synergistic Solutions Inc. announced additional platform support for **Smalltalk/SQL**, their portable database interface for Smalltalk. The product works in conjunction with the latest releases of ParcPlace Systems Objectworks/Smalltalk and Digitalk Smalltalk V. The product enables development of graphical user interface (GUI) applications that access information stored in relational databases. Direct database support is currently available for the Sybase, Oracle, NetwareSQL, and Gupta databases. DB2, Informix, Ingres, Rdb, and NetwareSQL databases may be accessed through a variety of gateway products.

GainMomentum

Gain Technology,
1870 Embarcadero Rd,
Palo Alto, CA 94303-3308
415.813.1800

Gain Technology, Inc. announced the general availability of **Gain*Momentum* Version 1.0**, an object-oriented multimedia application development and delivery system for medium- and large-scale multiuser computers. The software provides a complete environment for developing large-scale, interactive multimedia applications incorporating sound, full-motion video, animation, hypertext, graphics, and corporate data on corporate and departmental computer systems. It runs on the Sun SparcStation and SPARC compatibles. Versions for DEC, Hewlett-Packard, IBM, and Silicon Graphics workstations will be available by summer 1992.Gain developed its products under a joint-development agreement with Matsushita Electric Industrial Co., Ltd. of Osaka, Japan.

HI-SCREEN Pro II

Softway, Inc.,
China Basin Landing,
185 Berry St., Ste. 5411,
San Francisco, CA 94107
415.896.0708

Softway, Inc. announced the integration of PenDOS extensions into **HI-SCREEN Pro II**, its language-independent, object-oriented user interface development system. HI-SCREEN Pro II includes transparent support for the PenDOS operating environment developed by Communication Intelligence Corporation (CIC), providing access to advanced pen input capabilities without the underlying development complexity.

Send Product Announcements to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123, fax: (415) 928-3036. Include company name, address, and phone number.

tralization, downsizing, and open system initiatives within the organization.

The world changes

Organizations grow, merge, go into new business areas, develop new products, are faced with new competition, employ new technologies, and are subject to turbulent social, economic, regulatory, and political environments. Static models developed by the top-down information-engineering approach are unable to describe dynamically changing organizations.

Organizations are much more than just data

Organizations are people, resources, data, and processes that interact and behave in an ever-changing environment over time and space.

Barbara von Halle, in a recent article[1] found it was a startling realization to information engineers that data is perhaps only one-sixth of the information systems architecture necessary to promote data sharing in an enterprise. John Zachman's Information Systems Architecture framework, e.g., is composed of data (entities), functions (processes), network (locations), people, time (events/cycles), and values (business rules/strategies).

Data is not information

Information engineering enterprise modeling calls for the "normalization" (standardization) of the organization's data to reduce redundancy and improve the accuracy and consistency of the data resource. No attempt, however, is made to "normalize" the organization's processes or business rules. This means that different users can use different algorithms to process the data and can change or update data at different time intervals. Both of these activities can corrupt the value of the information retrieved from the databases. The flexibility of relational databases, the use of 4GLs, and downloading of database extracts onto microcomputer platforms all compound this problem as more and more people within the organization process data in different ways and at different times.

Direction

Most organizations do not have a strategic direction. Or if they do, senior management is not going to reveal that direction to the data-processing department.

Tony Durham indicates[2] that information engineers, in fact, complain that the difficult part of enterprise modeling, "is getting this woffle (sic) that comes out of senior management and working out what it means."

CRITERIA FOR EFFECTIVE ENTERPRISE MODELING

Overcoming these shortcomings will require enterprise models that are able to satisfy the following criteria.

Contextual

Systems theory describes the context or environment of a system as all of the influences on the system that the system cannot control. Organizations are shaped by many external factors, such as competition, the economy, government regulation, etc. All these influences and their effects on the enterprise must be identified and monitored. The most successful organizations are those that understand, respond to, and seek to control their environments. Familiar and widely reported examples of these organizations include American Airlines and American Hospital Supply. Both companies developed systems that allowed them to effectively understand, respond to, and control customers who were formerly an independent part of their environments. In addition,

> ***Static models developed by the top-down information-engineering approach are unable to describe dynamically changing organizations.***

there must be a way of "exchanging context" (i.e., understanding other points of view and other definitions of meaning) among all people and systems that will use the enterprise model.

Dynamic

Since the organization and its information processing needs are constantly changing, the enterprise model must also be constantly changing. While information engineers correctly realized that processes in an organization are not stable, they failed to understand that data is not stable either. Today, organizations are either changing or they are dying. New products and services are being developed continually; mergers, divestments, outsourcing, and downsizing are all dynamically changing the data and the semantics of the data used by the organization.

Semantic

Rich semantic models attempt to capture the essential features of reality needed to understand how the enterprise behaves and what services are required to support this behavior. A rich semantic model will, at least, have to capture information on the enterprise's data, processes, behavior, business rules, policies, resources, geographical distribution, organization, business events, and environment.

View linkage

Enterprise models must be able to be viewed from a number of integrated perspectives. John Zachman suggests that his Information Systems Architecture framework needs to be viewed from the perspectives of the business (the owner), the information systems (the designer), and the technology (the builder). Each of these perspectives requires different information, at different levels of detail, and in different contexts to describe the organization. In a recent article,[3] Zachman suggests:

> If your model-storage facility could maintain the integrity of each model (in his ISA framework), transformation algorithms between the models, versions of the models, and perhaps even versions of the entities within the models, it would provide an enterprise with the ability for dynamic infrastructure change.

View linkage, then, is the ability to dynamically link the perspectives and components of the rich semantic model. A change in the model entered from any perspective must automatically update all other views of the enterprise model.

Direct implementation

The enterprise model must be capable of being implemented without further work or investment. This means that the enterprise model will have to be a one–to–one operating model of the essential features of an organization, not just a collection of graphs, tables, and narrative. Direct implementation is necessary to allow modelers to prototype, validate, and use the enterprise model as the actual organization evolves.

Developed by users

The modeling process must be able to be performed by the enterprise's user personnel. There should be no need to have to involve technical personnel (who don't understand business processes) or change the model to accommodate physical DBMS implementation considerations (e.g., devising artificial intersection entities, normalization, denormalization, etc.).

OBJECT PARADIGM ENTERPRISE MODELING

The object paradigm offers an opportunity to develop enterprise models that satisfy these enterprise modeling requirements. Object paradigm enterprise models can be dynamic simulations of the organization. Such models will show the interaction of the enterprise with its environment. They will be unambiguous representations of the things and behaviors that make up the organization. These models will be rich, integrated, semantic representations of the organization that will be assembled and used by non-data-processing personnel who will be able to view the models from a variety of different perspectives. The following five major characteristics of the object paradigm will be used to implement this new approach to enterprise modeling:

Encapsulation

In the object paradigm, all things that exist in the real world are represented by unique, individual simulations called objects. Encapsulation means that the data attributes and data-processing methods associated with different objects are hidden from view. Once an object's data and methods are encapsulated into a simulation, the data and data-processing method can no longer be directly accessed. Instead, messages requesting various services can be sent to the object, which then responds (or behaves) by providing the requested service.

In the object approach, enterprise models will be developed by dynamically collecting the simulations that correspond either to the organization as it is today or to a series of "what-if" scenarios. These various simulations can be optimized according to the management's goals and critical success factors and then implemented in the real-world organization.

Extensibility

Extensibility allows the enterprise model to be composed of many different abstract data types. This means that the model can consist of video, graphic, audio, and even tactile representations. In effect, object paradigm enterprise models will eventually be "virtual reality" simulations of the organization.

Polymorphism

Polymorphism, the ability of new versions of an object to respond to the same requests for service, allows the physical implementation details of the enterprise model to be hidden from the user. This means that new hardware, software, communication devices, and even new business rules can be transparently incorporated into the enterprise model. For example, polymorphism allows the effects of changing business rules or even changing computer platforms to be hidden from the user, who still sends the same messages, e.g., "pay the employees" to the payroll object.

Class libraries

Class libraries are collections of templates used to create individual object simulations. Class libraries will soon become available for many objects that make up organizations. Electronic data interchange (EDI) and office document architecture (ODA) are two examples of initiatives to define and describe standardized business objects (e.g., financial instruments and office memos) that eventually

continued on page 13...

The processes on an ADFD are closely related to the information model object and the associated state-transition diagram, but they have an independent existence. A process may be shared by two or more ADFDs provided that it always consumes and produces the same data and events.

The information, state, and process models provide a comprehensive view of the components of an application system being studied. Shlaer and Mellor introduce the concept of *domains* to provide a mechanism for organizing groups of models according to the problem or service they provide. Common examples would include:

- user interface
- operating system
- network
- application subsystems

Individual domains communicate with each other using the client/server messaging model. Each domain is named and its purpose described in a mission statement that should provide a charter for constructing the OOA models. The resulting domain chart defines the system architecture for the complete project.

FUTURE DIRECTIONS

The book concludes with a discussion of project management issues, a language-independent design methodology (Volume three?), and an appendix that describes how to map the OOA work products onto mil-std 2167a reporting requirements.

UNANSWERED QUESTIONS

At the start of the review, I listed several questions that I wanted the book to answer.

First, how is the method described different than traditional SA/SD? The only major difference I could see is the replacement of the data dictionary with the information model. This is a significant improvement to SA/SD, but it does not constitute a new model of problem analysis.

Second, does the approach integrate the analysis of stored data and methods? To my mind, it doesn't. I was in general agreement with their discussions of the information model and most of the state model. The fundamental tenet of object-oriented technology is that behavior is paramount, yet the authors separate the storage of data from the computation on data. Why separate the accessor functions from the object? Why separate the event functions from the state? What is the justification for allowing a process to be shared by several ADFDs and directly access several objects?

Third, does the approach support the inherent concurrency of object instances? For the most part, yes. The techniques and suggestions presented provide a foundation for the analysis of dynamic behavior. This is not a cookbook, however, and additional reading will be required.

Fourth, does the approach maintain the semantics of the objects? The end of Chapter Three states that failure analysis is outside the scope of the book. For me this is the critical failure of the method described. An OOA methodology should support, even demand, that the semantics of an object be guaranteed at all times. The great hope is that O-O techniques will encourage widespread reuse of software components and enable distributed systems. How is this going to work if you cannot know or trust the behavior of inherited/delegated classes?

CONCLUSIONS

The real strength of this book is in the chapters on modeling complex dynamic behavior. The text and supporting examples clearly explain the relationship that exists between the information and state models, and demonstrate how these views should support each other. On balance, though, the book fails in its quest to define and articulate a new way of approaching software development. After reading the book, I have more questions than when I started. This is good in the sense that I am clarifying my understanding of object-oriented analysis but what I wanted was the "strictly defined, multi-level formalism" promised on the back cover. ≡≡

Calendar

Send calendar information to the Editor: Robert Shelton, 1850 Union Street, Suite 1584, San Francisco, CA 94123; fax: (415) 928-3036.

July 14–17, 1992 **Object Expo Europe** London, UK Contact: 212.274.0640	**July 21–23, 1992** **Object World** San Fransisco, CA Contact: 508.879.6700	**Sept. 21–25, 1992** **C++ By Design** New York, NY Contact: 212.274.9135	**October 18–22, 1992** **OOPSLA '92** Vancouver, BC, Canada Contact: 407.628.3602	**Nov. 16–20, 1992** **C++ World** Meadowlands Hilton, NJ Contact: 212.274.9135

The information model represents the persistent or stored data required by the application. To support the later modeling tasks, the information model must resolve all many–to–many relationships using associative objects and each object must have an attribute that maintains the current state of an object instance. The information model is comparable to an IDEF1X, information engineering, or CASE*METHOD entity-relationship diagram.

State model

The state model is used to describe the lifecycle of the objects defined on the information model. A state model consists of:

- *State-transition diagram*—documents the stages of an object's lifecycle, the events that indicate a progression from one stage to another, the actions needed to accomplish the progression, and the transition rules that define the legal progressions.
- *State-transition table*—an alternate representation of the state-transition diagram. Primarily used to verify that the state model is complete.
- *Event list*—a composite list of all events from all state-transition diagrams.
- *Object communication model*—documents the exchange of events between external objects (operators, physical devices, objects in other subsystems) and the state model.

A state model captures the common behavior shared by like objects. An object is an abstraction of a real-world thing. *What* we know about an object is captured by the information model in the form of attributes, relationships, and cardinality. *When* (at what point) we know something is captured by the state model in the form of events and the response of the object to those events. For many objects, the state-transition diagram simply records that something has happened (engines started, temperature reached, time expired, contract signed). For an associative object, however, the state-transition diagram must capture the rules governing the pairing of one object instance with another. The most interesting topic centers on the definition and description of an "assigner" state model. Its purpose is to define how contention between instances of one or more objects will be resolved (FIFO, Priority, etc.). This discussion raises an interesting side-effect of OOA. In traditional systems analysis, you are taught not to worry about discrete instances of a record or process. This is reasonable since a data value or software process is a static thing; it doesn't change from one invocation to the next. An object, by its very definition, captures and thus exhibits behavior. Therefore, during an analysis effort you must explicitly define how an object (a discrete instance) will respond based on its history and the history of any object to which it could react. The amount of detail you must deal with has just increased by (several) orders of magnitude. The book acknowledges this and provides a step-by-step procedure, suggestions, and model templates to help you mange this complexity.

Process model

A process model captures the functional aspect of the application system. It is based on Tom DeMarco's data-flow diagrams (DFDs) and is extended with control flows and conditional data flows. A process model consists of:

- *Action data-flow diagram (AFDF)*—documents the units of processing and the intercommunication between them. It shows the source and sinks of both events and information.
- *Object access model*—an alternate representation of the action data-flow diagram that highlights the synchronous communication between state models and object instance data.
- *State process table*—a structured text document that summarizes the set of action data-flow diagrams. The book recommends developing three copies; one sorted by process ID, one by state model and action, and one by process type.
- *Process description*—an unstructured text document describing the purpose of each process.

Unlike a traditional DFD, an action data-flow diagram is flat. Each state on a state-transition diagram is decomposed into a collection of separate units of computation, called a process, that communicate with each other using data flows and/or control flows. A process can also access a persistent data store (the ADFD representation of an information model object) or generate events to trigger a process in another state model. The authors also include a system clock and a timer facility that are used to synchronize the application system. Shlaer and Mellor classify ADFD processes as:

1. *Accessor*—`[Create]`, `[Read]`, `[Update]`, `[Delete]` of instances in a single information model object.
2. *Event generators*—produce a single event from the associated state-transition diagram. It is not clear from the text if each event is produced by only one process.
3. *Transformation*—computes a result based on its input data. A transformation object may directly read or write one or more information model objects.
4. *Tests*—a process that generates a conditional event or data flow.

Research or ready?, *continued from page 1*

3. Has the method been used on at least one prior commercial project that involved a team of five or more programmers?
4. Is the method supported by at least one reasonably priced (<$20,000 per programmer), commercially available CASE tool?
5. Is there at least one company with more than 10 people in it available to provide commercial training courses?

Commonsense screening, right? Yes, but after investigating the following methods:

- USA
  1. Booch from Rational
  2. Coad/Yourdon from Object International
  3. OMT (Rumbaugh) from General Electric
  4. Shlaer/Mellor from Cadre Technologies
  5. OOSD (Wasserman/Pircher) from Interactive Design Environment
  6. CRC described in the Wirfs-Brook book used extensibly by Knowledge Systems Corporation
  7. OSA (Embley) originally from Hewlett-Packard
  8. Odell/Edwards/Martin from Associative Design Technology
- UK
  1. SOMA (Graham)
  2. Fusion
  3. O-O JSD
- France
  1. Class-Relationship
  2. MOA
  3. OOM
  4. SYS_P_O
- Benelux
  1. HOOD
  2. OORA
  3. SDM-O-O
- Scandinavia
  1. ObjectOry
  2. OORASS
  3. OSDL
  4. EFSOS

only three methods passed these criteria. Nineteen others did not. Many seem a lot closer to the research end of the research/ready spectrum. Let's look a little closer.

APPLYING A COMMERCIAL SCREEN

Book screen

To pass the book screen, the method must be described in an English-language book. This is a nice objective criterion. Either an English-language book has been published and is available from technical bookstores or not.

Basic experience screen

To pass the author's experience screen, at least one of the authors must have participated in a commercial object-oriented development project, on a full-time basis, before they published the book. I was amazed by how many so-called "experts in O-O methodologies" fail this test. Many of the better-known authors had never participated in a single object-oriented development project.

Marie Lenzi, Editor of Object Magazine, seems to share my views completely. She describes it as "the famous person/famous company syndrome."[1] Being famous is not the same as knowing what you are doing.

Object-oriented projects are significantly different from traditional ones. The architecture, the design, the schedule, the deliverables, the development process, the tools, and the testing procedures are all different. New problems exist at each level. The learning curve on someone's first object-oriented project is so steep it is amazing that someone could propose a design method without having been through this process at least once. Most individuals in the industry who have project experience are busy designing systems while many of the software methodology authors are not very busy. Guess who started writing the books?

The diagram in Figure 1 tells the story as it exists today.

Designers with real experience in designing object-oriented systems are in strong demand as designers (fully booked). They can choose their projects. Their managers are extremely reluctant to let them take a vacation, much less a year off to write a book.

Experienced software methodologists have remained in fairly strong demand even as the technology was making a major change. This isn't surprising, as the lag be-

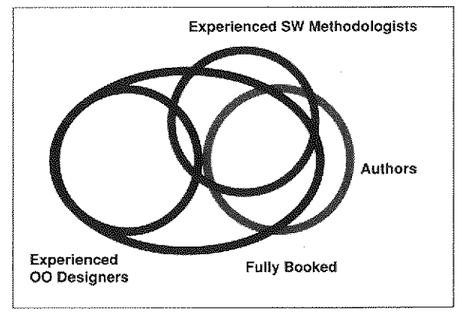


Figure 1.

tween the early adopters and the final adopters is so long and the market has become so large. Notice that there is very little overlap between methodologists and experienced O-O designers but the methodologists do tend to be authors. Percentage-wise, the methodologists are not in demand as much as the experienced O-O designers.

What we would like to find is a set of experienced authors who understand software methodologies and have been involved in several commercial O-O development efforts. They are few and far between.

COMPLETED PROJECT SCREEN

To pass the project screen, the methodology must have been used on at least one completed, hopefully successful, project. It would be even better if the methodology has been used on multiple projects. It would be better still if it had been used on a project similar to my next one.

A publicly available reference in the press is the most unbiased metric but that might be too severe a restriction. Even if it's privately obtained information, there should have been one project that actually used the method described in the book before the book was published.

> ***Serious O-O projects require allocating time and resources for design methodology, language, tool, and class library training.***

The question is, "can you provide me with a set of application briefs describing projects that used this method or tool?" Look for one that occurred prior to the most recent release of the book. Also look for one since the release of the book.

Application briefs give you the opportunity to contact some developers and managers who have tried the method to see what they thought. Questions you might ask include: Are subsequent projects underway? If so, will they use the method? What problems did you encounter and how were they overcome? What changes will you make on your next project?

Tools screen

Not all methods require automation. The CRC card method asserts that automation is inappropriate, at least in the beginning. To pass the tools screen, any portion of the method that would be unduly tedious if performed by hand should be automated.

The availability and cost of tools were factors in the early acceptance of both AI and CASE methods. To be sure we don't fall into the same trap, the tool must be available for purchase for under $20K per copy. That cost should include all software, such as a repository or encyclopedia, the user would require to use the tool.

Tools should be a released product by a company with at least one other software product. They should have periodic, scheduled maintenance releases and the product must run on at least one of the following workstations: Sun, HP, DEC, IBM, Apple, OS/2, or PCs with Windows.

Training screen

Training represents the most often underestimated portion of a project introducing new technologies. Most projects at least acknowledge that new language training should be scheduled, even if too little time is usually allocated. Unfortunately, many projects using new design methods only allocate money to purchase a few books and pass them around the project team members to be read.

Serious O-O projects require allocating time and resources for design methodology, language, tool, and class library training. To be sure training for initial and subsequent project teams can be obtained in a timely manner, the training should be available from a commercial training company of more than 10 people.

MY ASSESSMENT

I was surprised by how few of the most touted object-oriented design methods passed this rudimentary commercial screen. Only three methods passed; in alphabetical order, they are Booch, CRC cards, and Rumbaugh.

Booch

Grady Booch is the Director of Software Engineering Programs at Rational. He describes his methodology in a book, Object-Oriented Design with Applications.[2] Grady's work with Rational and its tools has provided solid object-oriented experience. Prior to writing the book, Grady participated in multiple projects including one on process control and another on telephony. Prior to this work, his initial experiences with objects utilized Ada and could be described as object-based rather than object-oriented. The process-control project mentioned above was written in C++ and consisted of hundreds of thousands of lines of code. The telephony project was written in Smalltalk and consisted of hundreds of Smalltalk classes.

Rational has a consulting division which regularly participates on projects using this methodology. Rational ROSE, a tool supporting the Booch method, is available from Rational. Experienced designers consultant, teach courses on the methodology, and provide tool training.

Book Review ≡≡

Object-Lifecycles—Modeling the World in States

by Sally Shlaer and Steven J. Mellor

reviewed by Michael Fuller

> *This is the second book by the authors describing an updated version of structured analysis techniques.*

The original book, Object-Oriented Systems Analysis: Modeling the World in Data described an extended entity-relationship diagramming technique and supporting documentation and introduced state and process modeling. The current book reviews the entity-relationship diagramming technique, describes updates and revisions introduced since 1989, and provides a detailed presentation of state diagramming and process modeling techniques.

OOA—WHAT IS IT?

The phrase *object-oriented analysis* (OOA) is frequently used but there is no consensus on its meaning and scope. Before beginning the book I listed several questions that I hoped the book would be able to answer:

- What exactly is object-oriented analysis?
- How does it differ from non-OOA?
- How does it differ from object-oriented design (OOD)?
- How does OOA ensure that object semantics are preserved?
- How does OOA integrate the specification of stored data and methods?
- How does OOA define and support object concurrency?

The book addresses each of these questions, but the answers may not be what you are expecting. Keep these questions in mind as we examine the book, and we will revisit them at the end of the review.

SHLAER/MELLOR OBJECT-ORIENTED ANALYSIS

The authors use three distinct models to describe an application system.

1. *Information model*—a Chen-style entity-relationship diagram that has been extended with subtypes.
2. *State-transition diagram*—a Moore-style diagram that represents states as boxes and transitions as arcs between the boxes.
3. *Process models*—a DeMarco/Mellor-style diagram that represents a process as a circle and information flow as arcs between the circles. An information flow can contain data (solid lines) or control (dashed lines) and may modify the information model (persistent store) or state model (dynamic store).

Each type of model is supported by text definitions and descriptions, and has a summary or overview graphical representation.

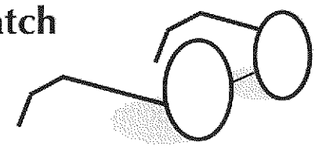
Information model

The information model provides the foundation upon which the state and process models are built. An information model consists of:

- *Information structure diagram*—documents the object types (class), its attributes, and the relationships in which the object participates.
- *Overview information structure diagram*—an alternate representation of the information structure diagram showing only the object name and relationships.
- *Object specification document*—a structured text that describes the objects and their attributes.
- *Relationship specification document*—a structured text that describes each relationship present in the information model.

Michael Fuller is an information engineering consultant and has developed large-scale, distributed applications with a variety of technologies including Eiffel. He can be reached at 415.928.7067.

Book Watch



The "Book Watch" column does not contain book reviews. These listings are abstacted from press releases provided by the publishers, and no endorsement is implied. Please send announcements to the Editor: Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123, fax: (415) 928-3036.

The Benjamin/Cummings Publishing Company (A division of Addison-Wesley), 390 Bridge Pkwy., Redwood City, CA 94065 415.594.4400

Benjamin/Cummings is pleased to announce the publication of Object-Oriented Design with Applications *by Grady Booch. The book is divided into three sections. The first introduces the basic concepts of the object model and explains techniques to identify classes and objects. The second introduces the author's method for object-oriented design, a graphical notation for the method, and includes practical advice on the challenges of managing object-oriented design projects. The third section is devoted to five realistic application projects, each of which is a complete design example implemented in a different OOP language: C++, Smalltalk, Object Pascal, CLOS, and Ada.*

Designing for O-O, *continued from page 15*

cept that allows me to physically enact modifications and see the repercussions against the model in real time via the SIB and BIS simulations described above. If I DELETE the major object as per our example above, I need to see exactly what happens even if it means that I watch my model destroy itself and my result is floating (orphaned) objects. The key here is the word *see*—it is imperative, in my mind, that my CASE tool be able to convey to me the *exact result* in real time. *The imagery is the solution.*

Let's look a little closer at the power of this interface. Much of my personal design work is done in scribbles, doodles, and sketches with my end users and the results are sketched on my office white-board during dialogue and debate with my design team. This interface, when married with the expert systems, must be able to act in the same fashion—as an intelligent white-board—and must provide me with the dialogue and feedback I need to create my model *on the spot*. It should be absolutely fluid, flexible, and responsive with none of this waiting for machine performance to stilt my creative thinking.

The tool should be something I can take to my users and sketch, in concept at least, their basic requirements, then automatically convey these to the rest of my design team. I need the coloured balls (classes of objects), the holes (characteristics and behavior), and the sticks (relationships and dependencies) in all three dimensions with some level of animation.

Then, and only then, do I have something comparable to those chemical engineers.

Implementing the models.

What to do on completion of this application model poses something of a problem. The designer would now have amazing technology with a superb picture of the design that is absolutely correct. This adds new meaning to the words *quality* and *completeness* in that the design is rigorous, meets the user requirements, and contains all the details required to fully portray not only the image of the object model but also the solution. The only output the designer should expect is a working application fully implemented in any language of choice.

I don't want to see the diagrams, I don't want to see the reams of paper documentation, I don't want to see the methodology, and I especially don't want to see the coded (language) application because, on achieving all the above, what we will really have constructed is a graphical language that will become as commonplace to us in the future as today's languages are now.

After all, in the last five years how often have you actually produced a core dump and decoded OCTAL or BINARY? ≡≡

CRC

There are several books and articles describing what has become known as the Class-Responsibility-Collaborator (CRC) approach. The approach was introduced at the OOPSLA conference by Ward Cunningham. Designing Object-Oriented Software[3] is the best-known book on the topic.

CRC cards, as they were first conceived, were hypercard stacks. Today, they are most often small index cards. Groups of people sit around a table. First, the group identifies the objects, their responsibilities, and then their collaborators. Next, the participants role play using the cards to test the dynamic behavior of the system under design.

One of the most valuable benefits of the way CRC cards are used is the ability to get everyone involved and stimulate group thinking. Computers are intrinsically single user. Trying to use the computers during these role playing sessions was found to be detrimental; thus, the lack of automation for this portion of the design process.

CRC is the most commonly used methodology for successfully deployed object-oriented projects.

Training for CRC cards is available from a variety of vendors. Several have made their own adaptations. ParcPlace includes the superclass on their cards; other companies have their own variations. According to Sam Adams, Knowledge Systems Corporation has taught the CRC design method to several hundred people in the past two years.

Rumbaugh

The methodology most often referred to as OMT is described in Object-Oriented Modeling and Design by James Rumbaugh et al.[4] Prior to publishing this book, the authors used object-oriented analysis, design, programming, and database modeling for several years on a variety of commercial GE applications. Their familiarity with both theoretical and pragmatic issues related to object-oriented concepts is apparent in the book. In addition, the book summarizes practical experiences on several medium-sized projects including some insightful lesson learned on each project.

Since the book was published in 1991, the methodology has continued to be used throughout General Electric, especially in GE Aerospace.

Object Oriented Analysis, a four-day course taught by GE Advanced Concepts Center, encompasses the GE Object Modeling technique described in Rumbaugh's book.

OMTool, a analysis and design tool which supports the object model in the method, as well as tool and method training is available directly from GE Advanced Concepts Center.

Research or ready?

The three methods listed are ready for commercial use. Other methods may soon be classified as ready, but most remain in the research category. Development managers beware.

A CHALLENGE TO OTHERS

During my research I heard a refreshing candor among the methodology providers. They are sincerely looking for methods that can help us all deliver commercial systems. One key individual raised the question "Do we need a formal methodology, or is that just a *passe* goal left over from traditional methods?" Others stressed the need for experienced designers to take the time to document their experiences.

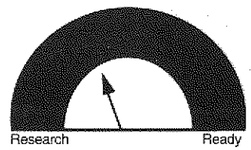
This article identifies the right types of questions for development managers considering a new O-O design method to ask. The methods that were not included in my assessment descriptions failed one or more of the criteria: book, author's experience, completed project, tools, or training. Some were a lot closer than others. I was told of many books in progress, projects underway, and tools that are close.

Hopefully this article will provide some additional motivation for the method providers to "belly up to the bar" and show us that their elixir has been tested on somebody other than their unsuspecting next customer. ≡≡

References

1. Lenzi, M. A. From the Editor, Object Magazine 2(1):8–10, 1992.
2. Booch, G. Object Oriented Design with Applications, Benjamin/Cummings, Menlo Park, CA, 1991.
3. Wirfs-Brock, R., B. Wilkerson, L. Wiener. Designing Object-Oriented Software, Prentice Hall, Englewood Cliffs, NJ, 1990.
4. Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. Object-Oriented Modeling and Design, Prentice Hall, Englewood Cliffs, NJ, 1991

Patti Dock recently joined OrgWare, Inc. as Vice President. She has been involved in the object-oriented marketplace since 1985 when she joined Stepstone as a technology consultant. Since leaving Stepstone, Patti has worked for Jackson Systems Corporation and General Electric, both of which are actively involved in the object technology. She currently teaches a course called OBJECTMethods that compares and contrasts leading object-oriented design methods. She can be reached at 203.270.1242.



STANDARDS

OMG's 18–24 Month View

by Christopher Stone

The Object Management Group (OMG) is dedicated to maximizing the portability, reusability, and interoperability of computer software and the business benefits derived from them.

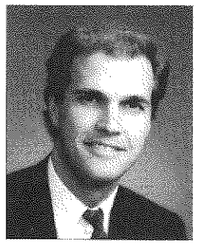
The OMG is the leading worldwide organization committed to creating a framework and supporting specifications for commercially available object-oriented environments.

The Object Management Group provides a reference architecture with terms and definitions upon which all adopted specifications are based. Implementations of these specifications will be made available under fair and equitable terms and conditions. The OMG will create industry standards for commercially available object-oriented systems emphasizing distributed applications development.

The OMG provides an open forum for industry discussion, education and promotion of OMG endorsed technologies. The OMG coordinates its activities with related organizations and acts as a technology-marketing center for information on object-oriented software.

Specifically, OMG is focused on:

- Providing object portability across heterogeneous systems.
- Providing interoperability of applications within a single object-management system.
- Supporting the design, analysis, and reuse of objects.
- Supporting the assembly of objects to form larger components.
- Supporting the trading and cross licensing of objects both between suppliers and users.



Christopher Stone is President of the Object Management Group (OMG). He may be reached at 508.820.4300.

PERSPECTIVE

Three years ago, OMG members and staff created an architecture called the *Object Management Architecture* (OMA). This model was meant to serve as the menu from which OMG would base its direction and build the interface specifications (see Fig. 1). The *Object Request Broker* (ORB) component of the architecture has been realized with the recent issue of the *Common Object Request Broker Architecture* (CORBA) Specification 1.1 (March 1992). In addition, the OMG Core Object Model is currently under development and should be published by July, 1992. The goal of this article is to outline the key *strategic* technology areas (not marketing) for OMG. Reader feedback is welcome. The instantiations of these technologies from OMG is in specification or application–programmer interface (API) form. They all support OMG's original mission and objective.

Object Request Broker

The Common Object Request Broker Architecture and Specification Revision 1.1 is OMG's first published API. Forty-five companies have agreed to support it so far. It describes the interfaces for accessing objects in a distributed environment long before distributed object applications have been written. An object (interface) definition language, a dynamic in-

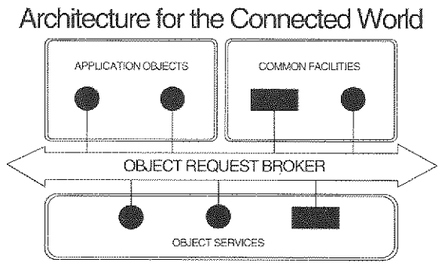


Figure 1. OMG's view.

Is is not logical, therefore, that we should expect expert systems to take care of the more mundane application of this predictability and not only guide us to completion of the model but actively provide the designer with an automated "assistant?"

Imagine a tool that is capable of responding like a PhD fresh out of college but with no real-world experience of application or database design—the dialogue would be invaluable. In fact, the analogy that springs to mind is that of a seasoned professor with a lab assistant avidly taking notes, notifying the professor of contradictions and anomalies as the dialogue and the object model ensued. The model could be completed with no redundancies, no contradictions, and no anomalies and the professor's perceptional awareness of the model would be significantly increased. Two minds are invariably better than one.

Again, given that we have captured the characteristics of the objects and their relationship to each other in the model the expert systems should be capable of "putting the model together." In other words, having defined our objects and given them characteristics and subclasses we should be capable of assembling the model via our requirements and dependencies on a "just do it" basis. The resulting model (always assuming there are no "holes") will be complete but may not be correct or valid.

> ***Imagine a tool that is capable of responding like a PhD fresh out of college but with no real-world experience of application or database design...***

ENSURING MODEL STRUCTURE AND VALIDITY

To ensure the correctness or relevance of the model to the requirements (or in applying change to the model), these knowledge-based systems have to be capable of simulating situations, applying them against the model, and feeding back to the designer actual results, based on the structural composition, those cascading actions have on the model. These simulated situations are the direct results of simulated transactions applied against the model *as if it were real*. Typically, they should be ADD(ed) or DELETE(d) elements extracted from a realistic scenario. Let me expand on this.

We have known for some time that the structural composition of the model will behave in predictable fashion under certain conditions—we call this structurally implied behavior (SIB). Similarly, we know that certain behaviours display predictable responses within known structures—we call this behaviourally implied structure (BIS). (The grounds for this knowledge may be the theme of a subsequent article on the subject of modelling theory but I would ask the readers' indulgence for the purpose of this article.) The marriage of these two aspects means that I can now apply this knowledge to my model to "shake out" the weaknesses, and make changes by looking at the effect, and reapplying simulations. For instance, the most common simulation I would wish to see is a DELETE against (all or) one of my major entities or objects inside the model. The expert systems should apply this in two fashions; first, a simulated destruct and, second, an automated destruct. Triggering the simulation of a manual deletion followed by all those that subsequently follow via the automatic deletion will tell me what effect I can expect across the entire model, predictable or otherwise.

This provides enormous power to actually see the effects in motion and respond to all areas of the model that give cause for concern. It also allows some other pause for thought. What happens if I DELETE the object EMPLOYEE from my corporate structure (data) model? If I see unexpected results, I merely modify the model to reflect the correct structure and resimulate against the model to ensure the correct response. If I see totally expected and correct cascading DELETES, then I have created an automatic archive feature into the model structure for the actual procedure for Terminating Employees.

Interfacing with the model

It is imperative in this kind of technology that we work at a much higher level than the current "logical model" space and so I will refer to this as a "conceptual model" space. Ideally, this is more flexible and fluid than the rigidly controlled, methodology-driven logical model of the past and it is imperative we look at the kind of interface that belongs on a tool of this kind. The interface, in fact, is key.

I need CAD-like imagery to display the model. I need the same tool that those chemical engineers already have including the concept of coloured balls (classes of atoms) with holes (valency) and sticks (bonds) to represent classes of objects, methods, and characteristics and I need these in three dimensions with full animation. What's more, I also want to create in software the con-

continued on page 16...

CASE ≡≡

Designing for object-oriented applications: a CASE for wishful thinking...

by Sue Hardman

Information engineers consider the core of any application design to be a set of multidimensional data and process models and yet persist in producing "blue print" model designs in schematic form.

I maintain that this is old-fashioned and inappropriate to the real world of designing complex business applications.

Imagine chemical engineers trying to construct a complex molecule using a schematic tool with joined box diagrams and a rigid set of diagrammatic rules to follow. Would they see the gist, shape, and form of the resulting molecule? Would it be appreciated that a DNA molecule was, in fact, a double helix? I suspect not. The chemical engineer of the 1990s has appropriate tools, meaningful colour-coded formal objects, CAD equipment with beautiful display technology, and the ability to construct, view, and animate not two but three dimensions. In return, the engineer actually "sees" his model.

Why, then, do we information engineers use outmoded, inappropriate formalisms and tools to express and view our application designs? Can we be expected to model adequately and realistically within the constraints of current methodologies and flat CASE technology or do our object models require the kind and quality of CAD tools that other engineering disciplines already have?

Sue Hardman has been a consulting applications designer for ten years, specializing in data and information engineering for optimum system performance. She runs the Service Operations group as part of the product marketing organization for the Desktop Division of COGNOS Inc., Ottawa, Canada. She may be reached at 613.783.6861.

CASE EXAMINED

The first wave of CASE tools automated standard, "structured" methodologies to produce schematic diagrams of application design models from either the *process model* or *data model* point of view. They were elegant to a degree but assumed fixed design patterns with laborious methodologies and construction rules for the designer to follow. They did not make for better designs. They provided some speed during the early design phase (but not enough to allow for interactive, creative thought to flow) and they assisted greatly in documenting our design layouts with a profusion of printed paper. Much was said about the "discipline" and "structure" of the approach but it remains questionable what the real benefit was.

Now we find the CASE marketplace has gone flat. Users have been disappointed by overstated promises of productivity and failure of the tools to respond quickly enough to the constant system and specification changes inherent in capturing levels of complexity. Many tools only addressed the problem of designing new applications; no one tool took on the problem of legacy applications and data.

I believe, however, that we are about to see a second wave of CASE products that will address the failures and weaknesses of the first wave with amazing new software technology, lightning performance, and the ability to engineer truly rugged object-oriented application solutions.

In fact, the CASE world may begin to look as follows.

DESIGNING OBJECTIVELY

Without doubt, these tools will be object oriented. The world of object orientation (like that of top-down structured) is one of structure, behaviour, classification, characterization, inheritance, and polymorphism that give us enough predictability on which to build the model. In other words, one object is differentiated from another by the characteristic traits we assign to it and it is these traits that dictate in what respect each separate object interacts and behaves in relation to other objects in the domain of the model. This is key to designing and engineering a given object's structure and behaviour. This predictability can be captured.

terface, and a read-only interface to an object repository have been established.

Interface definition language

The *interface definition language* (IDL) is the interface to the ORB core. IDL (including a dynamic interface) is a language binding meant to make a subsystem available from a given language. The only language currently supported is C,* although C++ lexical rules were obeyed with new concepts added for distribution. Portability of objects or applications among differing ORBs requires that applications built with class libraries support IDL; otherwise, these libraries will not be portable across all platforms. In addition, database systems have developed a standard *data manipulation language* (DML) called SQL. IDL, in database terms, is a *data definition language* (DDL). It is believed most computer languages, including DDLs (on relational and object databases) can be mapped to the IDL.

Objective: ORB interoperation—enable language-independent object interfaces (defined in IDL) achieve intra- and interoperability of ORBs.

1. CORBA 1.2
   - Clean up work on 1.1
2. CORBA 2.0
   - ORB to ORB interoperability
   - Federation (object sharing)
   - Full Repository
3. CORBA 1.X
   - Compliance test suite

Object services

Once a communications mechanism and single language are in place (the ORB and Object Model), a group of important, lower-level object services are necessary to make a commercially viable system. Some of these can be thought of as ORB extensions (e.g., security, transactions, persistence), while others are low-level features for ORB usability (e.g., configuration control, versioning, object linking and embedding). The list below reflects the present priority of services. This may change at the discretion of the Technical Committee (TC) and Task Force.

Objective: to add value to the Object Request Broker by providing distributed services.

1. Object Services Phase I
   - Persistence
   - Lifecycle
   - Repository
2. Object Services Phase II
   - Transaction processing/ concurrency control
   - Security
   - Naming/Events

Following this low-level service activity will be extensions to the ORB and Object Model work to define interfaces to common classes of applications (e.g., word processors, process control systems, spreadsheets). These are referred to as *common facilities*.

Object model

The most crucial component of the Object Services portion of the architecture is a common object model for describing object/class structure. The Object Model Task Force, active since mid-1990, has finished its draft model, based on a single structure intended for design portability only (the group will address source-level portability separately). A single model, with a separate component (feature) description and profiles (such as a model for ECAD, databases, distributed applications, etc.), has been accepted for publication by the OMG Technical Committee in the OMA GUIDE. The component and profile section of the core model will be done separately from the initial core work.

Objective: the Object Model defines a language-independent object structure and appropriate components and profiles. It will be used as a basis for defining design portability across all OMG specifications, related standards organizations, and ISVs.

1. Core Model—completed at May 27–28 meeting
2. Components/Profiles
   - Databases (ODMG)
   - Distributed Apps
   - Analysis and Design

Common facilities

OMG must define a standard base library of application classes and how they interact as an extension of the Object Services work. This will be comprised of definitions of specific application class interfaces for common applications (spreadsheets, word processors, etc.) always described in IDL. This work will lean heavily on the extant Open Systems Interconnect (OSI) document architecture standards (such as supporting the ODAconsortium), but expressed in ORB- and Object

* In addition, a structure must be put in place to solicit fast turnaround of specific language mappings to IDL. These proposals should be done through a Request for Proposals (RFP), no Request for Information (RFI) required. Targets would be OSF's NIDL, Objective C, COBOL, Smalltalk, and OSI's GDMO.

Model-compliant interfaces. These can be considered "high-level" class libraries. OMG has recently formed a Special Interest Group (SIG) to produce a survey of "low-level" class libraries (see description under "Class libraries") We expect this activity to begin in late 1992 or early 1993 with several request cycles lasting until late 1993 or mid-1994.

Objective: to ensure application portability and distributed development, the OMG TC should ensure close cooperation of the Class Library SIG and the creation of a Common Facilities Task Force with the main intent being to create a base set of small-grained objects (libraries) compliant with the Object Model, ORB, and parallel Object Services.

> ***Most commercially available class libraries are C or C++ based with little or no regard to the reuse and creation of distributed heterogeneous applications from them.***

SIGs AND DIRECTION

Special interest group activity

Although not directly in the path of standardization for distributed applications, the OMG supports several groups developing standards or informal positions in other areas. The call for this widening of interest has come from public awareness of OMG as "the place to go for objects," a perception that we have strengthened over the last year.

To avoid the losing our focus on the crucial distributed-application standardization efforts, however, we have (with the blessing of the TC) separated these activities from the primary TC activities into "Special Interest Groups." These semi-autonomous groups are working in a wide variety of object-oriented interest areas, some developing position papers and others actually preparing to establish standards. It is important to note that any OMG SIG publication, although they need not be accepted by the TC/Board route of adopted technology, are clearly marked as such on the cover.

Class libraries

OMG has recently formed a SIG to study the portability and interoperability of class libraries. Most commercially available class libraries are C or C++ based with little or no regard to the reuse and creation of distributed heterogeneous applications from them. To further complicate the issue, some languages (such as Smalltalk, Serius, etc.) include a base set of libraries from which to build an application. Others do not (C, C++). Given that the goal is application portability and interoperation, this SIG will develop a "survey" paper discussing availability and interoperation of low-level base libraries (i.e., list, string, array, etc.). The objective is to provide a set of base libraries for each language, ensuring interoperability. The libraries conforming in a CORBA-compliant manner would then be certified by OMG. If appropriate, a task force could be developed to issue standard interfaces for interoperation of these low-level base libraries in late 1992 or early 1993.

Object query

A major facility of any database management system, including object-oriented database management systems (ODBMSs) is an interface for querying the records of a database. The Object Data Management Group (ODMG) and ANSI SQL 3 are expected to be instrumental in providing assistance in this area as well as extending the semantic-based Object Model to include a syntax or "object SQL." They may also define the feature set of a low-level browsing tool to be available to OMA-compliant applications, for which the *data records* are application classes themselves. This will be a mid-1993 activity.

Analysis & design

Chaired by Andrew Hutt of ICL, this group is surveying object-oriented analysis and design methodology (CASE tools, etc.) for commonality with a view to drawing the diverse directions of this technology to some common approach or notation. This group has gathered considerable momentum and will issue a report comparing various methods with a reference model at the July, 1992 meeting in San Francisco. The group has set as its objective to create a task force in early 1993 to establish specifications for:

- repository model for object analysis and design
- tools for object analysis and design
- tools-interchange format.

Databases

The first of the OMG SIGs was created after the realization that all known object-oriented database companies are members of the OMG. This group, inactive during the Object Model work, will continue to develop standards of interest to object-oriented databases and object-oriented extensions to relational databases. This group has been transposed by the Object Data Management Group (ODMG) and has responded to the Object Services RFI with a single architecture. Rick Cattell of Sun Microsystems leads the group.

Smalltalk

Despite the long history of Smalltalk, no consensus-developed standard for the language exists. Duane Bay of ParcPlace Systems is pulling together a group within OMG to rectify this situation. A paper is expected in Quarter 3 of 1992.

End-user requirements

A first draft end-user requirements document, specifying requirements that impinge on object-oriented distributed applications, was developed by this group led by Mary Ostlund of AT&T in early 1990. Pat Davis of Boeing is a likely candidate to lead the completion of this effort in cooperation with Stefan Karlquist of Ericsson in Sweden. The visibility of this group will rise as more end-user organizations join OMG. OMG will also look for input from Corporate Facilitators of Object-Oriented Technology (CFOOT, chaired by Bob Marcus of Boeing) and the Forum on Distributed Object Computing (FDOC, chaired by John Rymer of Seybold Computing). Both of these are early user groups hopefully developing business scenarios and requirements.

CONCLUSION

With over 260 member companies and growing at a rate of five new members per week, OMG has positioned itself to bring dramatic change to the way we develop software. It is imperative that the user community begin to get involved in this process to provide both business and technical direction for OMG's members.

The next Technical Meeting will be held July 21–22, 1992, at the Sheraton Palace Hotel in San Francisco. This will be held in conjunction with the Object World Tradeshow held at the Moscone Center July 21-23, 1992. Look for future Object World shows to be held in 1993 in Boston, MA, and Weisbaden, Germany. ≡≡

Enterprise modeling, *continued from page 6*

could be simulated and stored in class libraries. Class libraries will allow users to model their organizations by requesting the services required by their enterprise from standardized collections of reusable simulations. Service requests will be directed to the appropriate class libraries by software agents called *brokers* or *traders*.

Reality modeling

In the object paradigm, enterprise modeling is the ongoing collection of simulation models that represent the real organization. When the organization requires new services or is called upon to behave in new ways, new objects or new versions of existing objects will be added to the collection. Reality modeling eliminates the need for the time-consuming attempt to model the organization from the top down. Instead, the enterprise model evolves over time by integrating all the object simulations that provide the services required by the organization. Put simply, the object paradigm enterprise model is the sum of all the classes of objects used to simulate the organization.

Reality modeling also means that the dynamic-simulation model can be used to replace current accounting and management reporting systems (accounting is, after all, just an attempt to represent real-world transactions) and control the real organization (e.g., when the model realizes it is the end of the month it could automatically initiate the payroll process).

CONCLUSIONS

The methods, techniques, tools, class libraries, and modeling notations necessary to apply the object paradigm to enterprise modeling are still under development. I intend to explore developments in these areas in future articles. The information-engineering approach to enterprise modeling, although well-intentioned, has some serious shortcomings. The top-down study of data, which may be useful to plan and design relational databases, is not going to provide a rich, dynamic understanding of the enterprise. Dynamic-simulation models of the enterprise, however, seem to be a promising opportunity for management to better plan, control, and operate their organizations. The models can be easily verified against the real world. Simulations that effectively provide the services required by the organization can be used and enhanced over time. Ineffective simulations can be immediately rejected at little or no cost. Object technology appears the appropriate vehicle to fulfill this promise. ≡≡

References

1. von Halle, B. The information systems architecture lightening bolt, DATABASE PROGRAMMING & DESIGN, January, 1992.
2. Durham, T. What's your line?, IBM SYSTEM USER, July, 30–34, 1990.
3. Zachman, J. Framework wisdom, DATA BASE NEWSLETTER 20(1):11, 1992.

≡ 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'd like to see but don't — or whatever other feedback you'd like to send. Send comments to Robert Shelton, 1850 Union Street, Suite 1584, San Fransisco, CA 94123; Fax: (415)928-3036; 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 (Kulinek)	❑	❑	❑	❑	❑
Feature (Adams)	❑	❑	❑	❑	❑
International Update	❑	❑	❑	❑	❑
ODBMS	❑	❑	❑	❑	❑
FYI	❑	❑	❑	❑	❑
Product Announcements	❑	❑	❑	❑	❑
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? _____

_____

_____

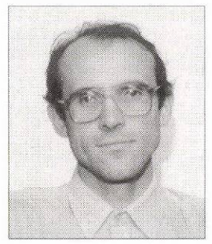
Name, title & phone (optional): _____

hotline on OBJECT-ORIENTED *technology*

VOLUME 3, NUMBER 10 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES AUGUST 1992

Object technology: toward software manufacturing

Object technology (OT) provides an opportunity for software manufacturing offering significant benefits other industries have enjoyed since the Industrial Revolution 200 years ago.



Eugene Kulinek

OT brings a fundamental change in the way we think of software. It is about building software products from existing software parts and building new software parts that can be reused throughout the corporation and the industry.

Today, most software is developed using a structured approach. Each application is an individually crafted product developed with little, if any, consideration for reuse. This is quite different from the approach of other industries, such as hardware development and car manufacturing, where building and reusing parts has a central place in building products. For example, we have parts builders such as Intel or Motorola and parts reusers such as Compaq or HP. It is hard for us to envision building computers from raw materials without parts.

Now, imagine we could build an application from available software parts. We could reuse small parts such as account and cheque or large parts such as foreign exchange or spell checker. We would also build new parts that others could reuse. This would lead to improved software quality, productivity, and flexibility with decreased complexity and cost.

This article discusses the transition from today's software development process to a new approach—software manufacturing. We will discuss required changes in software architecture and organization and some of the most important aspects of the changes needed to benefit from OT.

FROM CRAFT TO PARTS

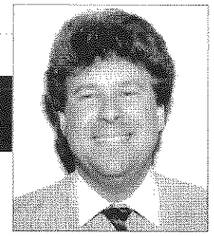
Object technology changes the paradigm from craft to parts. It requires a coordinated standardization effort, discipline, and commitment. It introduces new development processes, analysis and design techniques, and programming languages. It demands new ways of organizing development teams.

The following sections discuss the transition from today's software development process into software man-

continued on page 8 ...

IN THIS ISSUE

FROM THE EDITOR



In a previous issue of the Hotline, we posited the object of this technology: software manufacturing. Let's continue that conversation, keeping in mind a phrase from Mr. Sam Adams' column this month that "reuse on a large scale cannot be successful without...software components that are reuseful as well as reusable."

Our featured writer, **Mr. Eugene Kulinek** of the Royal Bank of Canada, takes the stand that object technology gives corporate and commercial software developers the opportunity to take advantage of software manufacturing, which he defines as building applications from off-the-shelf software parts. Manufacturing, Mr. Kulinek rightly observes, is making the move from craft to parts.

At a recent conference, Dr. David Taylor of Enterprise Engines used an analogy drawn from the manufacture of personal computers. The commercial PC can be understood as an assembly constructed from integrated circuits (ICs) and plug-in subassemblies. Subassemblies such as interface cards, power supplies, and memory banks are themselves constructed from integrated circuits. For most PC clone manufacturers, even the motherboard is a purchased subassembly. Discrete components are used sparingly, and only to support ICs.

The bottom line is *parts*—purchased, off-the-shelf parts. PC makers could build their own motherboards including application-specific integrated circuits (ASICs) to provide specialized features, improved performance, or built-in replacements for standard interface capabilities such as video display and networking. Most do not, opting instead to buy them from parts manufacturers. Those that do, reuse their ASICs!

Simple enough, we software developers say. Production computers are not built from discrete components. After all, what sane hardware developer would build a CPU from scratch? Most of us who have been in this industry for more than a few years have heard of (if not used) the PDP 11/70, the VAX 11/780, the Cray, or the PYRAMID (with a four—yes four—discrete board RISC CPU!). Performance and proprietary features were, of course, the driving factors in an era when the "microcomputer" was already a well-established concept.

What many of us don't remember, however, is the struggle to overcome the inherent drawbacks of custom design. I remember the delivery of early production VAX 11/780 systems, accompanied by teams of soldering-iron-armed technicians sent along to debug the CPU wiring. The machines did not, of course, work on arrival. They had to be debugged first!

Credit DEC with building most of their VAX 11/780 CPUs with ICs, though. The technology was still called LSI (large-scale integration). Each IC must have had at least 100 transistors in those days! Technology advances and manufacturing technology made possible the transition from low-level integration and custom-built, field-debugged processor boards to the wholesale use of commercial subassemblies and ICs. Here is where we reconnect with Mr. Kulinek's column. The software industry has left the period in its evolution where a reasonable argument can still be made for complete custom building. Sure, our organizations still need their ASICs but, as Mr. Kulinek points out, there should be people on our development teams whose job is to build ASICs, which will become corporate assets. As for the rest, what exactly is our excuse?

The *modus operandi* of today's corporate and commercial software developers should be to assemble commercial components, corporate ASICs, and off-the-shelf class libraries into finished assemblies (applications) that address business-user and product-customer needs. Let's get specific. What parts are available and how do you figure out which ones to use?

Today proven, high-quality class libraries can be purchased from many manufacturers to provide graphical user interface (GUI) services. Several provide what ParcPlace Systems Chairman, Dr. Adele Goldberg, calls "zero-cost portability." These products allow developers to assemble one application and deliver it to various combinations of Mac, DOS, MS-Windows, X Window System/UNIX, NeXTStep, and even OS/2 Presentation Manager without changing a line of application code. Such portability is available to any application de-

continued on page 3...

PARTNERINGS & ACQUISITIONS

IBM Corp. is acquiring an equity position in **Sapiens International**. The companies also announced a software development assistance agreement, the goal of which is to expand the set of application development tools supporting IBM's AD/Cycle. As part of the agreement, Sapiens will extend its client-server product, SAPIENS Workstation, to the OS/2 platform.

SunPro established operations in Europe, opening its European headquarters in Velizy, France. SunPro plans additional regional offices in the United Kingdom and Germany by the end of 1992.

Lucid, Inc. has named John DeArmon as Manager, Product Marketing. DeArmon will oversee product strategy and direction for Lucid's Energize Programming System. DeArmon will also manage marketing for Lucid's C and C++ compilers. DeArmon was most recently a software industry analyst for Dataquest, focusing on the computer-aided engineering (CAE) market.

Object Design, Inc. announced the formation of **Object Design Japan K.K.**, a wholly owned subsidiary in Tokyo, Japan. The new company, headed by managing director Michael J. Verretto, provides marketing and technical support for the company's ObjectStore object-oriented database management system (ODBMS) to its Japanese distributors and strategic accounts in Japan. Object Design also announced it will port ObjectStore to the newly announced UnixWare operating system for Intel-based PC platforms from Univel. The company has also joined Univel's Early Access Program.

Symantec Corporation and **Apple Computer, Inc.** announced a development and marketing agreement to provide a cross-platform application framework for Apple Macintosh computers and Microsoft Windows-based PCs. Symantec will provide the framework—the Bedrock framework—it is currently using internally to develop applications for Macintosh computers and Microsoft Windows. Symantec will leverage Apple's engineering resources and current object-oriented framework technology internally and work with Apple to support the developer community's transition to the Bedrock framework.

Ram Banin, a cofounder of **Daisy Systems**, now **Dazix**, is to become senior vice president of the Telecommunications group of Teknekron Communications Systems, Inc. of Berkeley, CA. At Teknekron, Dr. Banin will oversee the network management systems and object-oriented software businesses for telecommunications markets.

Siemens Nixdorf Informationssysteme AG (SNI) signed a licensing agreement for several of **Hewlett Packard**'s CASE SoftBench products: SoftBench, as it is incorporated in the Toolbus CASE environment from Informix; Encapsulator; and C++ Developer. SNI will incorporate the HP technology in DOMINO, SNI's CASE technology. In addition to the licensing agreement, SNI expects to become a sponsoring member of CASE Communique, a standards effort.

Interactive Software Engineering, Inc. (ISE) and **Versant Object Technology** announced they will jointly develop an interface from the VERSANT Object Database Management System (ODBMS) to ISE's object-oriented programming environment, ISE Eiffel 3. The interface will be developed in conjunction with ISE's European associate SOL of Paris. The VERSANT–Eiffel interface will be integrated into the EiffelStore persistent mechanism.

IBM Corporation will market the **Digitalk** PARTS product line through its Cooperative Software Program. The two companies previously announced that IBM would market the Digitalk family of Smalltalk/V programming environments.

Hewlett-Packard announced it will license its SoftBench Broadcast Message Server technology to software suppliers that want to create integrated software environments. HP uses this technology as the core of its SoftBench CASE product line. HP also said its Broadcast Message Server software will comply with the CAD Framework Initiative (CFI) specifications for intertool communication in electronic design-automation (EDA) environments.

HyperDesk Corporation announced that the HyperDesk Distributed Object Management System (HD-DOMS) will support UnixWare from **Univel**. Univel has provided early versions of its software to allow development of products supporting UnixWare.

Mercury Interactive Corporation and **CenterLine Software, Inc**. announced integration of their respective automated testing tool and programming environments. Mercury's XRunner, a robust automated testing system for The X Window System-based software, now interfaces seamlessly with CodeCenter and ObjectCenter, CenterLine's C and C++ programming environments. This announcement coincides with the unveiling of XRunner's new text-recognition feature.

CASE:W 4.0

CASEWORKS, Inc., 1 Dunwoody Park, Suite 130, Atlanta, GA 30338, 404.399.6236

CASEWORKS began shipping **CASE:W 4.0** for the Microsoft Foundation Classes (MFC), a code generator producing code for the MFC Library for C/C++ 7.0. In addition to MFC, CASE:W 4.0 supports Windows C API and Borland's ObjectWindows. CASE:W's snap-on capabilities eliminate the risk of selecting the wrong class library. Developers can generate code for the Windows C API or either of the class libraries. Once they have completed that work, they can "snap-on" a new knowledgebase and generate the code again in a different class library. CASEWORKS is also shipping CASE:W 4.0 upgrades for its C and ObjectWindows knowledgebases.

IRT capability

ICONIX Software Engineering, Inc., 2800 28th Street, Suite 320, Santa Monica, CA 90405 310.458.0092

ICONIX announced Integrated Requirements Traceability (IRT) capability in their multiuser, network-based CASE product **PowerTools**, allowing requirements to be incorporated into a CASE model and considered throughout the development cycle. Users can either copy requirements into a model from a source document or enter them as they are derived. Relationships between requirements can be entered and tracked and requirements can be allocated to appropriate elements of the model. At any point, one can trace requirements and verify that all have been considered in analysis and met by the design. Beyond documenting the accomplishment of project requirements, another major feature of the new capability is impact analysis, allowing the user to easily determine the impact of potential changes by reporting on every portion of the CASE model that will be affected.

C++/Views 2.0

Liant Software Corporation, 959 Concord Street, Framingham, Massachusetts 01701-4613, 508.872.8700

Liant's **C++/Views 2.0** enables programmers to develop portable applications for Microsoft Windows, OS/2 Presentation Manager, and UNIX/Motif without having to know anything about the specific rules and arcane structures of each windowing system environment. Programmers can develop software for one GUI and port the source code unchanged to any other environment by recompiling. Users are free to switch compilers and platforms as needed. Supported compilers include Borland 3.0; Zortech 3.0 and Microsoft 7.0 for MS Windows; Zortech 3.0 for OS/2 PM; and Liant C++ and all other standard UNIX compilers for UNIX/Motif. New features include a reengineered Notifier class that gives users full control of resource-based dialogs and support for multiple document interface (MDI). In addition, keyboard control of dialog support conforms to Microsoft Windows, standards for both resource-based and dynamic dialog boxes with complete support for accelerators. C++/Views includes a C++ class browser and source-code development tool, C++/Browse, which includes new editor customization features, automatic save of editing operations and restart of browsing sessions, ability to list all inherited member functions, and a streamlined selection of class dependencies.

Better-C V3.0, Top-Down Designer V3.0

Silico-Magnetic Intelligence, 24 Jean Lane, Chestnut Ridge, NY 10952, 914.426.2610

Silico-Magnetic Intelligence (SMI) announced **Better-C V3.0** program generator for C/C++ programmers and project teams and **Top-Down Designer V3.0** CASE design tool for C/C++. Among several enhancements, Better-C V3.0 boasts C++ code generation and Top-Down Designer V3.0 has been augmented to support object-oriented design specific to C++. The Better-C V3.0 package consists of tutorial, program generator, library source, and include files. The Top-Down Designer V3.0 package consists of tutorial and interactive designer software. Both require an IBM PC or compatible with DOS Version 2 and up.

TurboCASE, version 4,0

StructSoft, Inc., 5416 156th Ave. SE, Bellevue, WA 98006, 206.644.9834

Version 4.0 of TurboCASE for the Macintosh, from StructSoft, Inc. offers full object-oriented support for encapsulation, inheritance, and polymorphism. 4.0 supports five new editors, four of which are graphics editors that create different class diagrams. The fifth editor, a dictionary, gives the user the ability to define classes. New TurboCASE diagrams include Class Hierarchy, Class Definition, Class Collaboration, and Class Design, all integrated through a project database, providing multiple views of the software design. All information entered in a diagram is automatically recorded in the Data Dictionary, which eventually becomes the design specification. TurboCASE supports most widely-used methodologies including Shlaer/Mellor for object-oriented analysis.

Quest ObjectViews C++ 3.01 (OVC++)

Quest Windows Corporation, 5200 Great America Parkway, Santa Clara, CA 95054, 408.496.1900

Quest Windows Corporation announced the release of **Quest ObjectViews C++ 3.01 (OVC++)**, a comprehensive, object-oriented user interface development environment for the X Window System based on and 100% upward compatible with the InterViews toolkit. OVC++ features full support of OSF/Motif and OPEN LOOK including dynamic support for X resources: event translation, menu mnemonics, menu accelerators, predefined dialog boxes, option menu, scale interactor, paned window, file selection box, OL push-pins, Motif tear-off menus, and full key-board traversal. OVC++ offers a complete set of development tools including ObjectBuild (an object-oriented C++ graphical user interface builder), OVC++ class browser, periodic table of OVC++ objects, sample demos, and source.

...continued from page 2

veloped for the OBJECTWORKS/Smalltalk virtual machine by design. NeXT and OS/2 are to be delivered soon. Smalltalk/V currently supports MS-Windows and OS/2, and will shortly support X/UNIX. For developers outside the Smalltalk world, such libraries provide capabilities unavailable with your raw development environment. Another group of class libraries that can be purchased for Smalltalk dialects, C++, Objective-C, and Eiffel, among others, provides truly object-oriented views of databases in relational database management systems (RDBMS). Given the momentum behind Object COBOL, we should expect class library products for it shortly, as well. NeXT bundles a public-domain compiler on their systems (available from the Free Software Foundation) that will compile any combination of class libraries and traditional functions from ANSI C, C++, and Objective-C. Manufacturers like SUN and Hewlett-Packard are also delivering development environments that allow developers to take advantage of class libraries written in languages other than their native one.

Could most of us see financial and delivery-time value in such components? Can we agree that development environments are rapidly breaking some of the language barriers by providing enhanced—and often portable—virtual-machine capabilities? Do our projects require us to build our own GUI and database classes, or could we use commercial products already available? When we really stop to look at the commercial components available today, is there anything we are really going to do so much better that it will justify the total cost? So why do we allow developers to continue to build our applications from scratch? Whose responsibility is it that our organizations don't have a reuse culture?

I will leave this thought for consideration and return to the second question I raised earlier.

In his first column of a four-part series, **Mr. Sam Adams** of Knowledge Systems Corporation addresses the question of how to determine which classes and class libraries to use in assembling an application from parts. He suggests that understanding a problem (analysis) or developing a solution (design) can best be done through interactive role-playing. The class–responsibility–collaborator (CRC) approach he describes can help take developers and users out of the computer realm and empower them to act out and feel the problem or solution domain.

Extending this construct to software manufacturing, one could easily substitute off-the-shelf business ASICs and software ICs where a particular class was needed—at any level of granularity from Mr. Kulinek's spell checker class to a "simple" window or database class. As several of this month's authors point out, responsibility for making this come about lies with developers and managers. This is all about creating a reuse culture. In upcoming issues, Mr. Adams will show us various aspects of using and extending the CRC concept. This is only one proven approach to understanding and designing that does not require practitioners to buy particular brands of computer-aided software engineering (CASE) tools. It supports reuse and validation of the class selected for reuse through interpersonal simulation.

Where Mr. Adams is headed is toward software asset management: developing and buying components and subassemblies that increase in value over time through reuse and safety. The first benefit is fairly obvious. If part `WindowClass` costs $500 and is used once, it is less valuable to our organization than if it is used 100 times. The cost per use is $500 and $5, respectively, with approximately the same overhead each time for selection and validation (in other words, it really does cost you something to make the decision that `WindowClass` is the right part to use, and regardless of cost, we have to consider it). The second benefit is less obvious to people involved in new development with limited support and maintenance experience. If you buy and use `WindowClass`, and I (on a separate project for our company, perhaps) build my own `MySuperWindowClass`, both have to be maintained. Users would have to be trained to recognize and use each of our window objects. Changes to improve one would probably lead to a demand to improve the other, which increases the likelihood of failure—an event that will probably occur just when the CEO is watching someone use this nice new application that she just authorized $750,000 to build! Maybe, if we're really lucky, the failure would occur just as my paycheck was being processed...then, and only then, would I learn?

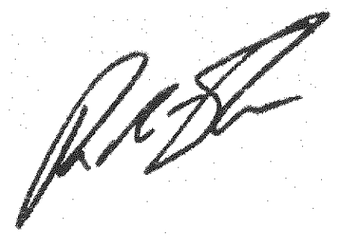
Ah, yes, it all comes back to money. Maintenance is generally regarded as 80%-plus of the lifecycle cost of a software component, so reuse would avoid 100% of the cost of something whose real (lifetime) cost is not completely evident to the developer. Were I to reuse your `WindowClass`, 50% of the total cost (as well as uncounted collateral damage) could be avoided—if for no other reason than the commercial part you purchased had already been well debugged by yourself and hundreds or thousands of other customers. That's leverage! That's software manufacturing.

Mr. Daniel Levin of SERVIO Corporation, Japan, writes of objects and oat bran—sort of. The Japanese are *importers* of software technology, especially when it comes to object technology. Beware, he cautions—Japanese culture promotes the critical value needed for reuse: humility. The simple assumption that I would probably not build a better `WindowClass` than the one built by a parts maker specializing in GUI technology changes my entire attitude and approach toward software development. Personal value is placed on using someone else's better idea rather than on making my own just to "be all that I can be."

Mr. Levin also promotes iteration in the product-development cycle. This reminds me of Dr.Goldberg's warning about the difference between protocycling and iteration, an important difference here. Like Dr. Goldberg, Mr. Levin talks about an organized whole process that starts with a clear goal and a quality development process (such as Total Quality Management), and then rapidly improves the product through extensive customer feedback. Change is the norm, but this process differs radically from protocycling, where iteration becomes an excuse for undisciplined hacking. The former can be measured; the latter cannot. An iterative approach to design (such as CRC) also makes possible component reuse, where Mr. Levin suggests the Japanese have the rest of us flat beaten!

Mr. Sergiu Simmel of Oberon Software takes a look at some of these concepts in the context of managing persistent objects. Keep software parts in mind as you read his description of the Kala architecture. Then consider efforts like the Common Object Request Broker Architecture (CORBA), Open Software Foundation's Distributed Computing Environment (DCE) and Distributed Management Environment (DME), and the concept of micro-kernel architecture that underlies OSF's effort to provide a common virtual machine across platforms that look as different to their users as do UNIX, HP MPE, and IBM MVS. They are all about building assemblies from subassemblies and parts.

Although it won't be easy, our focus must be on two concurrent tasks: make parts and use parts.



RETURN ON INVESTMENT

Software assets and the CRC technique

by Sam Adams

In most large organizations today, software is a large financial liability. Hundreds of millions of dollars are spent annually in an attempt to extend the life of millions of lines of brittle, patchwork code.

Sam Adams is the Senior Consultant and cofounder of Knowledge Systems Corporation. Since 1984, Mr. Adams has been actively developing object-oriented software systems in Smalltalk and is widely recognized for his expertise. He is codeveloper of the group facilitation technique using CRC cards and has been training computer professionals in object-oriented technology for over six years. Mr. Adams has served on several conference committees and is a frequent speaker and panelist at leading industry conferences. He can be reached by phone at 919.481.4900 or by fax at 919.460.9044

The situation is expensive in the short term and intolerable in the long term.There is a consensus among computing-dependent organizations that something must be done, and soon.

Most large organizations today are aware of the often miraculous claims made about object technology (OT). Many have decided to integrate the technology into their businesses. The question for these organizations is not "Do we go object-oriented?" but "How do we maximize the benefits of object technology and manage the risks?"

Answering these questions will be the focus of a series of articles, of which this is the first, concerned with meeting the challenges of enterprise-wide computing using OT. The following topics will be covered:

- Defining the requirements for maximizing return on investment in OT.
- An overview of the CRC technique and how it can help a business create reusable software assets.
- Extending the CRC technique to address the complete software lifecycle.
- Multiuser tool requirements for the deployment of object technology on an enterprise scale.
- Managing software assets using object-oriented metrics.

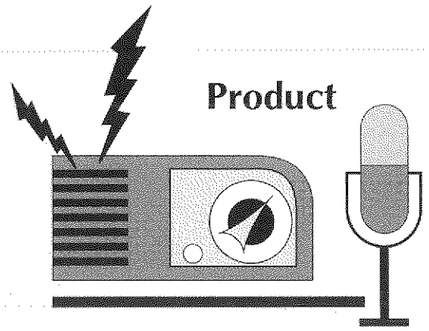
OOP AND ROI

KSC's experiences over the last six years helping large organizations adopt object technology have shown that maximizing the return on investment in OT requires a new set of assumptions:

1. Software is a corporate asset. As an asset, it has a value that can appreciate through investment in its quality and reusability. This value is enhanced when a software component can be reused in many different applications.
2. Pervasive reuse of high-quality software components must become the norm. A reuse-based infrastructure is the single most critical success factor in meeting the ever-increasing challenges of software development. But reuse on a large scale cannot be successful without the existence and proper management of large libraries of software components that are "useful" as well as reusable.
3. The most valuable software assets of any organization will be the objects capturing the essential nature of their business domain. High-quality design information, not just code, will be the foundation of these assets.

ACHIEVING SOFTWARE QUALITY

High-quality software meets or exceeds the needs of the user without violating user expectations. Achieving this level of quality requires that the designers of software focus on the needs and expectations of the user. Traditional software development processes assume that quality can be "tested in" after the software has been developed. The user produces a specification, the development team does their best to develop the application, then the software is tested and patched until it is either accepted by the user or abandoned. That many software projects are abandoned due to poor quality is ample proof that a different approach to software quality is required. At the highest level, our approach is based on three principles:



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.

SoftBench 3.0/ C++ SoftBench 3.0
Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304 303.229.2255

Hewlett-Packard introduced several new software-development products designed to help reduce programming complexity, increase performance, and facilitate parallel development of complex applications. **SoftBench 3.0** and **C++ SoftBench 3.0** are distributed software-development environments based on the SoftBench framework. Also released are **C++ Developer 3.0**, a class-construction and browsing tool included in the C++ SoftBench environment, and **Encapsulator 3.0**. Programmers can encapsulate window-oriented programs into their development environments via the Terminal Object with Encapsulator 3.0 without source-code modification. Also released is **ChangeVision**, a software change-request management environment that helps automate and manage software change requests on UNIX-based computers. It collects, analyzes, and correlates software measurements such as code complexity, defect density, test coverage, and schedule status and provides software teams with insight into the status of their projects. **SynerVision** for SoftBench, which enables users to construct computer-aided process-management environments, was also released. Software-development teams can attend the **ChangeVision Metrics Workshop** to assist them in using that product. Also offered is **HP Software Inspections**, a three-day workshop, based on a process used by HP, teaching managers and developers how to perform software inspections and integrate inspection data with reports generated by HP's ChangeVision software. Attendees learn how to set up pilot programs and implement a software-inspection process across their organization.

Smalltalk/V for Windows
Digitalk, Inc., 9841 Airport Boulevard, Los Angeles, CA 90045, 310.645.1082

Digitalk's new version of **Smalltalk/V for Windows** includes support for Windows Multiple Document Interface (MDI), a ToolPane (a row of buttons that perform functions when selected), a StatusPane that displays information on the status of applications, an ObjectFiler for sharing objects easily with other applications and developers, HelpManager support, support for non-US character sets, and performance improvements, as well as source-code browsers, inspectors, and push-button debuggers. It provides interfaces to Dynamic Data Exchange (DDE), allowing information to be shared between Smalltalk/V and other programs; and Dynamic Link Libraries (DLLs), providing a mechanism for calling code written in other languages from within Smalltalk/V. This release, Version 2.0, takes advantage of new features in Windows 3.1 while maintaining compatibility with Windows 3.0. Smalltalk/V source code is compatible with Digitalk's Smalltalk/V programming environment for OS/2, allowing developers to develop on either platform and deliver applications on both systems simultaneously.

Consulting
Enterprise Engines 4008 Bayview Ave. San Mateo, CA 94403 415 573-0363

Taylor Consulting announced it is expanding its operations and has incorporated as Enterprise Engines Inc. David Taylor is Chairman and CEO of the new company. Bill Morton and Mike Jarrett, formerly with Informix and other high-tech companies, have joined the company as President and Vice-President, respectively. Enterprise Engines Inc. will help companies scale object technology up to the organizational level, building active information systems known as enterprise engines. These software engines confer two major benefits: They reduce system development costs by reusing models of standard business components and increase white-collar productivity by including workflow, simulation, animation, reasoning, and other advanced capabilities into the business models.

O-O analysis and design module
Popkin Software & Systems Inc., 11 Park Place, New York NY 10007-2801 212.571.3434

Popkin Software & Systems, Inc. is shipping an enhanced **object-oriented analysis and design module** for System Architect, their PC-based CASE tool that provides support for Booch 91 and the Coad/Yourdon object-oriented techniques. Booch 91 includes design support for systems being developed for Ada, Smalltalk, Object Pascal, C++, and other object-oriented languages. The Coad/Yourdon technique uses a new notation to show five layers of models: subjects, class and objects, structures, attributes, and services. In addition, as part of the core product, System Architect includes support for the Schlaer/Mellor methodology.

Objective-C extensions
Berkeley Productivity Group, 35032 Maidstone Court, Newark, CA 94560, 510.795.6086

Berkeley Productivity Group's Borland **extensions to Objective-C** make it possible to port Objective-C to Microsoft Windows without changing a single line of code. The complexities of windows memory management are concealed from the Objective-C source code. Standard input and output can be performed to a Windows window. Alternatively, the Objective-C software can be run as a message server interfaced to spreadsheets or Smalltalk. All preprocessor, compiler, linker and postlink switches required for compatibility between Objective-C and the Borland compiler are given and the reasons for them explained.

Send Product Announcements to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123, fax: (415) 928-3036. Include company name, address, and phone number.

Predictions

Software programming, long considered more art than science, is becoming a disciplined endeavor as object-oriented technology becomes the linchpin for provocative computer applications. Integrated software, networked knowledge and digital money are just a few of the applications that will be swept forth by an advancing tide of object-oriented technology. Indeed, according to Wayne Rosing, president of Sun Microsystems Laboratories, object-oriented software will be a key enabler of the distributed networks that will emerge in the last half of the 1990s...At the same time, many of tomorrow's applications will build on—but move far beyond—the concept of OO programming. In doing so, they will take advantage of the truly distributed computing environments of the future. The most interesting of those applications are likely to revolve around new methods for disseminating knowledge as a service on the network of the future. "The next big gain is going to be re-encoding knowledge in a much more active form and then being able to provide a marketplace for that knowledge," Rosing said. "Conventions will emerge for how knowledge gets represented in rich, complex ways so people can manipulate it for their own use." Objects will play a key role in those new ways to represent knowledge. Manipulation will most probably be accomplished using hypertext-like search capabilities. Most important, the idea of integrated applications will be prominently featured...To pay for such services, a common coinage will emerge called digital money....

Multiple applications, Alexander Wolfe, ***ELECTRONIC ENGINEERING TIMES****, 5/18/92*

...Over the next couple of years, software agents on the new wireless computing devices will be "semi-intelligent," and able to help users with such applications as calendaring and email, but plagued by a tendency to circumvent pro grammers' intentions by taking instructions too literally, [according to Apple Computer's Alan Kay]. In the second half of the decade, he predicts that the agents will become increasingly smarter. At the same time, the rise of object-oriented programming will let end users start to build more of their own applications, in much the same way they now create their own spreadsheets and work processing documents. Already, there are more HyperCard programmers in the US than Cobol programmers, and many of these HyperCard programmers are end users, he pointed out. Further, even now, OOP software is allowing users to perform such tasks as modifying email systems....

"Intimate computing" is the wave of the future, Jacqueline Emigh, ***COMPUTER CURRENTS: SAN FRANCISCO BAY AREA****, 4/21/92*

Applications

In the past year, faculty and staff members at Allegheny College have written more than 100 sophisticated computer programs that are being used in classes. The programs were developed by about 25 people, most of them faculty members not highly skilled in the arcane art of programming...Most institution that install public computer networks use machines made by the International Business Machines Corporation—or less expensive clones—or manufactured by Apple Computer Inc. That, says Joel M. Smith, assistant professor of philosophy and director of educational computing services at Allegheny [College], is because those two different types of "platforms," as they are called, have the largest variety of software already written for users...As a result, he says, campus administrators and technical experts choose platforms for the educational software that already exists. The problem is that while most professors will gladly use a broad application for such tasks as word processing, few instructors like using instructional software written by someone else for their own courses. That is because the program rarely meshes with their teaching styles...But most professors don't have the technical expertise needed to write a program, so they must work with computer programmers. That approach, says [Carnegie Mellon University's Robert] Sheines, has big problems, as well. "Programmers don't know anything about pedagogy, and professors don't know anything about computers"...Specialists at Allegheny have been developing a special library of objects that can be used in educational computing, including objects that represent Petri dishes, bar graphs, and tables. Once the code for each object is written, other, less sophisticated programmers can easily include it in their own applications....

College enables professors to write computer programs with ease, David L. Wilson, ***CHRONICLE OF HIGHER EDUCATION****, 5/20/92*

Standards

...The biggest shortfall of current object-oriented DBMS technology is the lack of a standard definition and manipulation language. This limitation results in a dearth of data modeling facilities and standard content modules for OODBMSs....

On the shoulders of giants, Mark R. Jones, ***DATA BASE PROGRAMMING & DESIGN****, 5/92*

Analysis and Design

...Programming can become so easy that customers will "program" without even realizing it. That's when the real payoff comes—when ordinary people put multimedia information objects together into presentations, training materials, reference materials and so forth. "Programming? Who, Me? I'm just writing."....

Workstations—a software advantage?, Nick Arnett, ***MULTIMEDIA COMPUTING & PRESENTATIONS****, 4/30/92*

1. Maximize user involvement throughout the process. As the ultimate judge of the quality of the software, the user is the best person to determine if his needs and expectations are being met.
2. The right decisions must be made at the right time in the process. The earlier a design decision is made during the process the greater its impact on system quality. If an implementation decision is made too early, it can lock out potentially better design alternatives. If a business analysis decision is deferred too long, it may require massive changes to the design and implementation or simply cost too much to include in the system.
3. Expect and encourage iteration throughout the software lifecycle. The best way to insure a high-quality result is to exploit the hindsight developed during the project as well as the foresight gained from previous projects. Only with continual design validation, measurement, and refinement throughout the process can constant quality management be achieved.

MAKING THE RIGHT DECISION AT THE RIGHT TIME

To know *when* to make the right design decisions, we must take a fresh look at the process of software development. Traditional approaches separate analysis, design, implementation, and testing into discrete steps, all insulated to some degree from each other. While it is necessary to divide large development projects into deliverable phases for management purposes, divisions along these lines produce many undesirable side-effects. The frequent criticisms of such "waterfall" processes bear witness to this fact.

We consider the development process to be a continuum that begins when we first attempt to understand the domain of a particular business problem and ends when a successful, high-quality implementation is put into practical use. Design, from this perspective, is the continual process of discovering, evaluating, and deciding between alternatives at all levels beginning with initial domain object discovery and ending with the validation of the last line of program code.

Early in the design process, business rules and other domain issues predominate. As the initial design evolves, other issues, such as legacy-data integration and client–server distribution, become the main issues driving further extension and refinement. Later, the focus shifts to implementation issues such as execution efficiency and memory utilization. All of these are design issues and all require design decisions. A major benefit of this perspective is the lack of barriers between these levels in the design. At any time, an innovative design alternative may be proposed and evaluated at any level. This encourages iteration and contributes to a higher-quality result.

Application modeling and behavior distribution

In the design of all applications, whether interactive or batch, a major design decision involves the division of labor among the entities involved in the system. These include not only individual software objects but also humans as interactive users and external systems such as mainframe-based legacy systems or manufacturing work-cell-control interfaces.

It is important to model not only the interfaces between these entities but their individual responsibilities in meeting the requirements of the application as well. This is difficult in most methodologies, since they primarily focus on the design and implementation of software objects only and tend to leave the description of external entities to associated documentation not integrated with the method itself. To fully understand and meet the requirements of today's complex applications, the behavior of both internal and external entities must be determined and modeled. This includes the human–computer and client–server interfaces, both of which are typically neglected in object-oriented methodologies.

Focusing on behavior and interaction

The challenge in finding a way to consistently model these widely differing entities is evident. We have found, by modeling entities in terms of their behavior and interaction, that both internal software objects and external entities can be represented in such natural ways as to be accessible to non-computer professionals like users and domain experts.

There are three major benefits to this approach:

1. As humans, we have decades of experience dealing with complex systems of interacting entities that come complete with their own behavior. In fact, this is main reason why design techniques like CRC cards are so successful: they let us apply our natural skills and experiences to the field of system design.
2. By focusing on behavior and interaction, we can defer more implementation-oriented decisions such as whether to store an attribute as data or compute it dynamically until a more appropriate level in the design process.
3. By limiting the design of the system to the entities, their responsibilities, and interactions, we make a much larger part of the design process accessible to the system specifiers and eventual users, one of the keys to highquality mentioned previously.

INTRODUCING THE CRC TECHNIQUE

CRC is an interactive technique for one or more people that uses note cards and role playing to facilitate the process of modeling a system in terms of objects, their behavior, and their interactions with each other and

with entities that lie outside the system under consideration. The technique was originally developed by Ward Cunningham while at Tektronix, Inc. in the mid 1980s to help him communicate object-oriented designs to nonprogrammers.

CRC stands for class, responsibility, and collaborators. The information written on a CRC card (Fig. 1) consists of the name of the object (actually, the name of its class), a list of its responsibilities written as concise, active verb phrases, and a list of the other objects that collaborate as service providers to assist this object in the discharge of its responsibilities.

The technique itself is a simple one. Candidate objects are proposed and each object's potential responsibilities and collaborators are explored. Scenarios are developed to validate the design against system requirements and then tested using roleplaying. During roleplaying, the responsibilities and collaborators of existing objects are extended and refined and other objects are created as needed to complete the design. Existing scenarios are also refined to reflect changes in requirements and new scenarios introduced as system requirements are added. This process continues iteratively until the design can successfully complete all the scenarios and thus meet all system requirements.

The result of the process is a set of object descriptions captured on cards and a set of scenarios that demonstrates the behavior of the system and validates it against a set of requirements.

The importance of roleplaying

As mentioned above, the CRC technique focuses on behavior and interaction of objects and other entities. Roleplaying, or acting out the behavior of the objects, is a powerful tool for the refinement and testing of object designs. While most methodologies rely on diagramming notations to attempt to capture and communicate complex interactions between objects, role playing allows the designers to actually experience the behavior firsthand. This theatrical anthropomorphism has many benefits in the design process. Since designs can be "executed" very early in the process using scenarios, alternative designs can be explored easily using roleplaying as a form of rapid prototyping. Designs as complex as entire manufacturing systems can be simulated in surprising detail, taking advantage of the temporal and spatial nature of roleplaying that can be only poorly captured on paper. When used in a group setting, roleplaying provides a mechanism for the "parallel processing" of design problems since each "player" need only focus on the behavior of his or her own object instead of everyone in the group attempting to understand the entire system individually. An additional benefit of roleplaying in design groups is that it tends to help involve everyone in the design process, regardless of their background or experience, so all participants can add their unique value to the process.

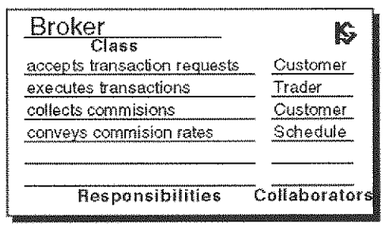


Figure 1. A typical CRC card.

An example CRC design session

Let's look at a partial CRC design and refine it using roleplaying. This design is an enterprise model for a mail order company that sells products to customers using catalogs and credit card orders taken by phone or mail. Some of the necessary cards are shown in Figure 2.

Example scenario

The scenario we'll use for this session is one where a Customer, "Joe," decides to purchase two pairs of wool socks and some climbing boots from the Backwoods Outfitters, a mail order company. He calls a Salesperson, "Carl," at Backwoods and places his order. Carl attempts to confirm the purchase, only to find that Joe's credit card is at the limit.

Example roleplay

For the roleplay, let's skip to where Carl is about to attempt the purchase confirmation. As he collected the purchase information from the customer, Carl created a Purchase object and described the product orders: two pairs of wool socks and one pair of climbing boots. He also told the Purchase object about Joe's credit card. We pick up the roleplay after Carl asks the Purchase object to run a credit check on Joe's credit card. The players include the objects Purchase, ProductOrder (2 instances), and Product (2 instances):

Purchase
includes ProductOrders for wool socks and climbing boots.
knows about Joe's credit card

ProductOrder 1
Product: Wool Socks
Quantity: 2

ProductOrder 2
Product: Climbing Boots
Quantity: 1

Product "Wool Socks"
Price: $20

Product "Climbing Boots"
Price: $120

F.Y.I.

Excerpts from leading industry publications on aspects of object technology

Hybrids

...Indeed, in line with the growth of networked and client/server computing, new object-oriented tools have emerged on workstations and PCs. And though many have characterized that development as a threat to CASE—and a key reason behind the less-than-stellar sales for AD/Cycle—others also see a gradual and complimentary merging of more traditional mainframe-environment CASE tools and object-oriented and minicomputer CASE tools. The I-CASE procurement at DoD—potentially worth billions of dollars—embraces a grand vision of software development. At the heart of that vision is the central repository with common interfaces into which all the standards-based vendors can plug their products. Templates and objects extracted from past systems would become the standard pieces that developers would use to develop systems for everything from accounting to inventory tracking.

CASE grows up, Andrew Jenks, ***WASHINGTON TECHNOLOGY****, 4/23/92*

...If you want to move to object-oriented technology, you have got a couple of options: You can unlearn what you have done and go off in a pure object-oriented venue, or you can think in terms of extensions, like C++ is to C. Our view is that there are hundreds of thousands of people conversant with 4GL technology and they really have the job of writing new applications and maintaining the old. Given our organic view, we want to add object-oriented extensions to the existing 4GL. A similar extension is to add a graphical environment to the suite of tools that 4GL utilizes. We are doing both of those kinds of things with 4GL. Companies providing developers with all new, object-oriented environments, are saying that it allows them to escape from yesterday's world and move forward. But a truer view is that you are telling them to adopt the object-oriented metaphor by itself and trapping them within what that world provides. I think being able to marry the two can very often give you the best of both worlds, rather than a compromise of both worlds....

Framework for the future, Chuck House, ***OPEN SOFTWARE JOURNAL****, vol. 5/ issue 1*

Strategies

...The real demand for OO technology will explode when companies start to tackle the next, radically different generation of applications, said [Al Fung, of SoftFab International]. "Up until now, we've been forced to adapt to computers' limitations," he explained. "From now on, users are going to expect computers to adapt to them, through applications that are easy to use. But developing easy-to-use software requires a tremendous amount of work, and that's causing a crisis in software development. The only way to manage the complexity is through object-oriented development."..."An object-oriented system can have an object that represents the CEO and an object that represents the vice president of sales, and these objects are self-contained entities that have attributes just like the people do. That more closely models reality, which translates into making things easier to deal with." In time, said Fung, that sort of approach will be extended through all aspects of a system, so that software developers will be able to construct, enhance or modify systems as easily as a Windows user can copy a file. "The overriding goal of all software development efforts should be to make things easier for humans," he said. If OO-based approaches can do that, then what's everyone waiting for?

The objective approach, David Freedman, ***CIO****, 5/15/92*

Products

...The smartest thing Steve Jobs and his team did was to come up with its object-oriented software. Steve and the gang should be very worried about Gain, which is licensing its software to NeXT's competitors, including IBM, which clearly has abandoned its past interest in NeXTStep. The threat to NeXT is that Gain's objects aren't just for building interfaces, they extend all the way down into programming and information objects. What's worse, for NeXT, its much larger arch-rival, Sun Microsystems, is going to bundle Gain-based multimedia applications with Sun workstations....

Workstations—a software advantage?
Nick Arnett, ***MULTIMEDIA COMPUTING & PRESENTATIONS****, 4/30/92*

...Consider Apple's plans for System 8—a microkernel that will run on both 68000 and PowerPC CPUs. This microkernel is supposed to add such modern OS conveniences as hardware memory protection, preemption, multithreaded task management and other goodies. Great! But how about adding built-in object support for a new version of HyperCard? Five years ago HyperCard was hailed as a breakthrough in user programming, combining object orientation with hypertext and graphics. As Apple develops System 8, a New HyperCard should be ready to be proclaimed as another breakthrough for people trying to get control of their computers. Anything less is unacceptable.

What's in the cards for HyperCard's future?
Don Crabb, ***MACWEEK****, 5/11/92*

time of writing (of course there are more now). My slight reservation would be the overemphasis on Ada-restricted techniques such as HOOD, GOOD, and MOOD. Whilst the Ada design community has contributed significantly to more general object-oriented ideas, I could not recommend any of these methods to an industry embarking on the challenge of true object orientation. Although the chapter focusses on OOA/D, there are still moments when implementation and language issues unfortunately encroach, e.g., stating that "the traditional object-oriented principle [is] that objects are specified by their methods alone"—true in implementation, not true in analysis, as the author then goes on to show.

One of the more interesting parts of the book is also the most original: the introduction of the author's new methodology: SOMA (Section 7.3.2). This elaborates upon the Coad and Yourdon work, utilizing several AI-derived notions, and provides for interesting reading. Section 7.5 concatenates CASE tools and lifecycle models, which I think deserve more discussion; first, in separate subsections and, second, in a subsection discussing the synergism rather than this simple juxtaposition. Again, product information on CASE tools given here will be well appreciated by industrial adopters.

Chapter 8 begins to answer the much-asked question "What are the management implications of adopting an object-oriented lifecycle?" The answer given is basically prototyping; although the need for deliverables and milestones is also stressed (p. 294).

The final chapter attempts the impossible: to forecast the future. This is hard to do in general and even harder in software engineering. Topics covered include language trends, expert systems, AI, uncertainty, open systems, concurrency and parallel processing, formal methods, hardware concerns, and, finally, a discussion aimed at management on how to really begin using object technology. Graham sees a window of opportunity about to open (Fig. 9.2) and urges industry "to adopt object-oriented methods cautiously but quite wholeheartedly;" I concur with this soundly based optimism.

With a few caveats, I can recommend this text primarily because it is important that object technologists do not become ostriches and ignore both the technical and managerial lessons available from AI and, specifically, expert systems, similar to those we are currently learning from the data modelling community. ≡≡

Customer
read catalogs — Catalog
select products — Catalog
purchaseproductusing credit — MailOrder Company, CreditCard

Product
convey description
convey unit price
convey accessories — Product
convey quantity on hand — Warehouse
convey reorder information

Salesperson
accept purchase requests — Customer
acquire purchase information — Customer
confirm purchase — Purchase

ProductOrder
convey ordered product — Product
convey order quantity
determine total order cost — Product

Purchase
convey customer — Customer
convey method of purchase — CreditCard
determine total price — ProductOrder
convey product orders — ProductOrder
determine tax — TaxSchedule
check customer credit limit — CreditCompany

CreditCompany
issue credit cards — CreditCard
maintain credit limits — Customer
approve credit purchases
determine interest on credit
bill customers — Customer

ProductList
convey product offerings — Product
maintain product offerings

Warehouse
reorder products — Product
store products
convey product quantities

Figure 2.

To run a credit check, the Purchase object must know the total cost to the customer including tax. To determine this, each ProductOrder is asked for its own total cost. To determine its total cost, each ProdutOrder asks its Product for its unit price, then multiplies it by the order quantity. Summing the totals from each ProductOrder, the Purchase object now has the total cost before taxes. To determine the tax to add to the cost, Purchase needs to consult the TaxSchedule object, but while it has been referenced as a collaborator on the Purchase CRC card it has not yet been defined. So, we add a card for TaxSchedule:

TaxSchedule
determine sales tax — Purchase

Now that the Purchase object can determine the tax, it can determine the total cost to the customer and run the credit check by calling the CreditCompany and getting approval. We immediately recognize that the CreditCard object has not been defined yet, so:

CreditCard
convey customer name
convey credit company contact info — CreditCompany

Now the Purchase object can contact the CreditCompany and get approval. Upon doing so, we find that the CreditCompany requires the account number, for which our CreditCard object is not currently responsible, so we add that responsibility:

CreditCard
convey customer name
convey account number
convey credit company name and phone number

Having consulted the CreditCompany, Purchase finds out that Customer Joe's credit is insufficient for this purchase, and informs Salesperson Carl in reply to his original request for a credit check. Carl then informs his customer, and the scenario is complete.

SCALING UP CRC FOR THE ENTIRE LIFECYCLE

The CRC technique has been proven to be an effective approach for high-level object-oriented analysis and design from single applications to large-scale enterprise models. The technique has been accepted across the O-O industry and is currently in use in different forms by the majority of practicing O-O analysts and designers worldwide. What else is required to address the broader needs of the entire software development lifecycle? There are two major areas not addressed by the CRC technique in its original form. The first is system requirements acquisition and analysis. The second is system implementation and testing. The next article in this series will deal with these issues, as well as provide an overview of a complete lifecycle methodology that is centered on the CRC technique and extended to provide integrated support from requirements to code, and back. ≡≡

Object technology, *continued from page 1*

ufacturing, touching on the implications for both software architecture and organization.

Craft

Craft is the way we develop software today. We usually acquire some base software such as libraries and an operating system. Then we engage in a number of independent projects to obtain products from each undertaking, as shown in Figure 1. The thick lines between each project represent the walls existing among them—there is no, or at best very little, reuse among projects. Today's organization and reward structures encourage such behavior.

Parts

The parts approach offers two relatively independent software streams (see Fig. 2): software parts and application development. The mandate of the software parts stream is to build reusable software parts whereas that of the application development stream is to build applications using these software parts.

I suggest some changes to the above development process that will open enormous potential in software manufacturing. First, let us build applications themselves as parts. Applications would break down into several interworking parts. A typical word processor, e.g., could break into a spell checker, thesaurus, and editor. We would actually have no applications but rather a set of software parts interworking to accomplish specific tasks.

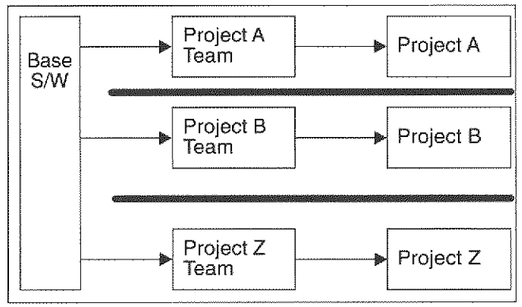


Figure 1.

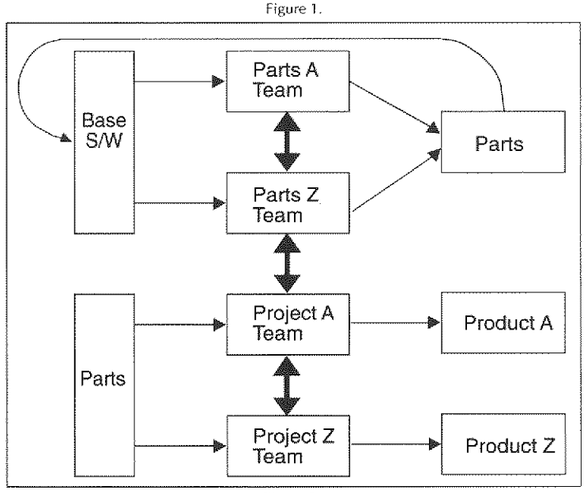


Figure 2.

As a result, we would not have *application* development teams—every team would be involved in developing various reusable software parts (see Fig. 3). This is very similar to other industries, e.g., the automobile industry, which can reuse spark plugs or mufflers (simple parts) and engines or automobiles (complex parts).

The parts approach lends itself to the development of distributed computing environments where some objects (usually complex) act as a servers to client objects. For example, both `branch banking` and `trader` objects (clients) would use a `foreign exchange` object (server) to perform some exchange related services on their behalf. Both clients and servers could reside on different, heterogeneous machines. Adding additional clients or servers or tailoring functionality to specific requirements is also significantly easier.

Architecture

In today's typical computing environment, there are a number of applications running concurrently. Although there may be some limited sharing among applications (usually through shared data), most applications are designed to be self-contained and do not share their functionality with other applications (see Fig. 4). Thus, we may have a word processor with a great spell checker but another application, say, a spreadsheet, cannot use it—it needs its own.

Object technology supports complex software parts (e.g., spell checker, editor, spreadsheet, GUI) being defined more independently of a particular application and thus reusable by anyone. Our object-oriented spell checker could act as a server to a number of different clients such as a spreadsheet (Fig. 4). Obviously, such an approach requires standardization of communications among software parts—something very natural in other engineering fields but, unfortunately, not in software.

Fortunately, there is some hope on the horizon. The OMG is working on a set of standards for interworking complex software parts in a distributed environment in which parts can communicate using the Object Request Broker (ORB)—a software bus. The approach is similar to computer hardware that uses a bus and plug-in boards. We can purchase a modem from any vendor and, as long as it conforms to an agreed standard, integrate it with the rest of the computer functions.

But what if we want to build our first application using OT today? For various reasons (lack of an ORB for our target platform, lack of corporate commitment) it is un-

Book Review

Object-Oriented Methods

reviewed by Brian Henderson-Sellers

by I. Graham
Addison-Wesley, Wokingham, UK, 1991
410 pp

This is an interesting and different introductory approach to object-oriented ideas.

It has, as the author readily admits, a stronger artificial intelligence (AI) and database (DB) bias than many other competing texts. This has the twin effects of introducing new ideas into the O-O community from "sister" software engineering subcultures and, conversely, of sometimes leaving one feeling that one is studying AI/DB without a clear indication as to how it relates to O-O. In other words, the synergisms could be stronger and more clearly obvious.

Graham notes the tendency for subcultures in software engineering to keep reinventing the same concepts under different names. This book does a lot to encourage the merging of these areas: objects, expert systems, and data modelling.

The book's strengths are in its wide and varied coverage and its practical slant. The obvious belief of the author that these ideas are not only exciting but applicable to industry now permeates throughout. That the author himself has an industrial background gives credibility to these claims. The book's weaknesses in my view are some of the imprecisions and potential confusion in the early discussion of basic concepts. Whilst I concur strongly with the author about using a single concept (I call it object/class, he calls it object) as a unifying feature of the whole lifecycle, I found myself in disagreement early in Chapter 1. For example, there has been discussion lately on the confusion between terms like *abstraction* and *classification*, *encapsulation* and *information hiding*, etc. Graham uses the terms abstraction and encapsulation interchangeably, which adds a new slant to the terminological confusion. A few pages later he tells the reader that encapsulation is equivalent to information hiding; also that an abstract data type (ADT) is similar to a class—it is certainly associated but it is generally regarded that an ADT relates to the specification and a class to the coded object module and therefore an ADT is the equivalent of specification augmented by implementation. On page 14 we are told that "the terms encapsulation, data abstraction, information hiding and message passing all refer to much the same thing."(!)

The book has nine chapters, an appendix on fuzzy objects (not recommended by the author for the faint-hearted!) and a useful glossary (bearing in mind my concerns about definitions raised above). The citation list is good, as are the name and subject indices. Each chapter ends with both a succinct and very useful summary and an annotated bibliography/further reading section.

Chapter 1 introduces basic concepts; Chapter 2 discusses many of the software engineering attributes of object technology first discussed in detail by Bertrand Meyer. Whilst the author focusses on concepts rather than languages, there is a digression into languages in Chapter 3. Here the section on functional languages seems, to the O-O novice, to be out of place; not because it necessarily is but because it is not obviously integrated into the remainder of the chapter. Again the author's acknowledged interest and background in AI is evident and provides new understanding of the interrelationships that we (the object community) still need to explore. This AI theme continues as an undercurrent in Chapter 4, "Applications," again containing material I have not seen in other O-O introductory texts.

Chapter 5 discusses database technology in the absence of objects as a prelude for the addition of objects in Chapter 6. This covers both theory, products, and industrial applicability in a very useful 28 pages. This is followed by the largest chapter (87 pages) on analysis and design. The author quite correctly (in my view) avoids an arbitrary split between analysis and design to create two smaller chapters. It discusses in useful detail a good many of the object-oriented analysis and design (OOA/D) techniques that were available at the

Brian Henderson-Sellers is with the School of Information Systems, University of New South Wales, Australia. He can be reached at brianhs@usage.csd.unsw.ozau.

Finally, the application objects are implemented in the upper layer(s), resulting in an application set or single application. They are implemented using the concrete, object-delivered services provided by the layer immediately below.

KALA—A BUILDING BLOCK

The layered architecture sketched above has been the goal for developing Kala: the persistent data server from Penobscot Research Center.[1–3] Kala implements the untyped data layer in Figure 1. Most of our effort during the research and development program leading to Kala has been in compressing the diversity of functionality required by object management environments into a suite of small, compact, reusable primitives.

Two fundamental insights have guided us:

1. what's common across *all* persistent environments is applicable to persistent data made out of nothing but uninterpreted bits and references into other persistent data, *and*
2. what's common across *all* models for transactions, versioning, concurrency control, access control, licensing, configurations, and security is scope-based visibility management.

By reducing all services to a handful of primitives, we have abstracted the commonality among all useful environments without imposing either a least common denominator or a most comprehensive solution. Instead, Kala is a tool for building such environments to fit specific needs while preserving the lowest-level interoperability.

This approach to creating a functional layer devoid of any object or data model is hardly new. In the same domain of storing data, an iterative process in the 1960s and 1970s led to the contemporary notion of a file as a stream of uninterpreted bytes, which in turn has served as one of the most pervasive *de facto* points of interoperability. If the industry had continued to insist on the structured files common in early operating systems, our ability to even pass a simple text file between two word processors would have been much hampered.

Kala carries the notion of a semantics-free layer further by adding several areas of functionality including the visibility management primitives, full software recoverability, references, and active (executable) data, etc.

So how is this different than contemporary object-oriented database management systems (ODBMSs)? The fundamental difference is packaging: there are certain kinds of functionality that make a product a database management system (DBMS). These include a data or object model, e.g., relational, object-based; a data manipulation language (DML), e.g., SQL and, in modern ODBMS products, object-oriented 3GLs such as C++; a data definition language (DDL), e.g., nonstandard relational schema notations or, in modern ODBMS products, the declarative sublanguage of 3GLs such as C++; and a suite of tools for navigation, query, schema management, etc. Kala has none of the above; thus, it definitely does not qualify as a DBMS—nor does it attempt to do so.

Instead, Kala is a medium-grain building block: a reusable component. Its approach to the functionality it provides is different, too. Instead of covering a laundry list of functionality, Kala provides combinable primitives used to produce not one but many such laundry lists.

The approach is similar to that of many programming languages: they don't provide sorting as a language statement but provide other primitives one uses to write one or many sorting routines, and then one uses the primitives to write something completely different, like a payroll program, thus effecting reuse at the language facility level.

In summary, the level of specialization is considerably higher with DBMS products than it is with Kala. This comes not only from functional versatility but also from physical characteristics such as size and efficiency. Kala is not a replacement for a DBMS but an alternative attractive to those whose true requirements are not quite fulfilled by DBMSs. In the realm of object management systems, Kala presents the opportunity to provide for economically attractive commonality while supporting the inherent and practical need for diversity.

CONCLUSION

The layered approach proposed by the Kala architecture brings modularity from the object level to system-level granularity. This contrasts sharply with many architectures that still yield monolithic products with one single usage at a time. Layered OMS architectures provide interoperability without compromising user functionality and nonfunctional qualities (such as performance, compactness, etc.). ≡≡

References

1. Simmel, S. KALA—INTERFACE REFERENCE PART I: KALA FACILITIES, REVISION 2.0, SOFTWARE VERSION 2.1, Penobscot Development Corporation, December, 1991.
2. Simmel, S.S. KALA MAIN CONCEPTS, Penobscot Development Corporation, 1990.
3. Simmel, S.S and I. Godard. The Kala basket—a semantic primitive unifying object transactions, access control, versions, and configurations, PROCEEDINGS OF OOPSLA'91, October, 1991.

likely we would take the OMG approach to its full extent. However, we should design and develop our systems with it in mind: build our applications of complex parts that can be made independent when the OMG approach becomes feasible in our organization, separate service objects (GUI, communications, database) from business objects (account, client), and build independent complex parts such as foreign exchange, branch, or spell checker. The separation of functionality is also essential to reusability.

Why we craft

Why is the majority of software today still crafted despite the obvious advantages of building from parts? I think the main reason is that it is very easy for someone to begin building a single software application. All we need is an editor, compiler, and one average programmer (and short-term view). It is also easy to duplicate written software with a simple diskcopy command. Those attributes do not apply in many other engineering fields, e.g., building tires or transistors, where we need a significant undertaking to build a single part and it is rather hard to diskcopy it. This is why these industries thought of parts at a very early stage.

However, building software is not much different from building Boeing 747s if we take a *long term view* of *all* our software applications that should be integrated together. Imagine we were told to develop not one but ten different banking applications. I am sure we would think of reusing the efforts from one application in another and would want to create some common software parts before building our first application. We would see that a large coordinated group effort was preferable to ten teams running independently. As in the 747 analogy, it would not be possible to diskcopy our first application to build the next one. Unfortunately, most software today is built with a short-term goal in mind—delivery of a single application.

INTRODUCING OT

Significant changes are required to introduce OT into an organization. The following sections discuss the most important factors of the change.

Organization

Object technology requires a different organization of software development. Most important is the existence of two sets of teams, with separate mandates, that have to be managed differently:

1. *Software parts teams* with a mandate to develop and maintain reusable software parts (business, graphics, integrating with the third-party class libraries). These teams work with business experts to formulate reusable business models and concentrate on developing frameworks and abstract classes. The teams own the software parts.
2. *Application teams* with a mandate to develop applications reusing those software parts. In cases where parts do not exist, these teams coordinate development of necessary parts with the software parts teams. They provide feedback about the quality of the parts they reuse but do not own them. This calls for a different attitude than that of today's application development teams—they no longer own the world.

For OT to succeed, a commitment from senior management is essential. The change is too significant to be left for developers and technical managers to attain. For example, management has to reinforce and reward reusability (both creating reusable parts and reusing available parts) in all departments developing software. Availability of reusable classes is vital to the success of OT in an organization. The software parts must be counted as corporate assets of strategic importance.

To minimize risk, OT could be introduced in a care-

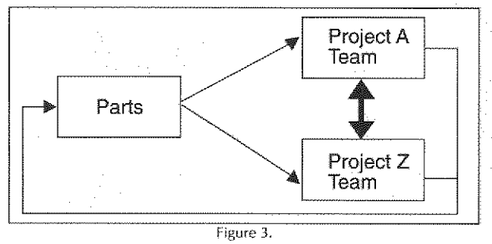


Figure 3.

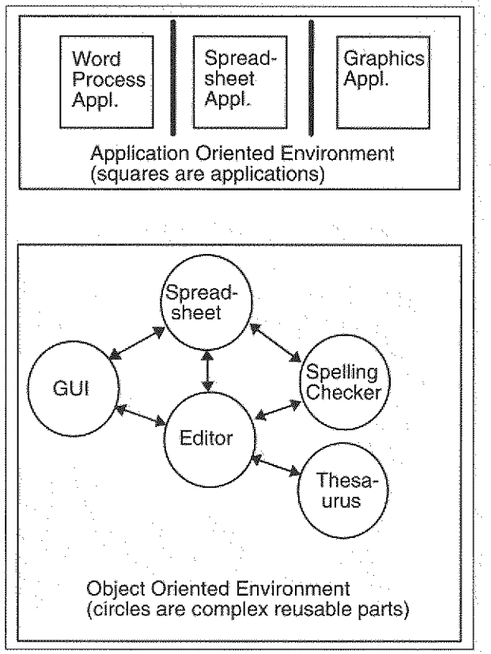


Figure 4.

fully selected part of the business. In banking, this could be in a branch operation. As we build more reusable parts we can expand to include other business areas. With each expansion, we revise and refine our business model. Another benefit of this approach is that we will experience the effect of reusability much sooner because, with the limited resources we usually have, we will make better progress in building parts in a smaller section of our business.

Development

Object technology brings new methodologies into the development process and requires different thinking. Instead of data and functions, we think of objects providing services and their relationships with one another. Message parameters and attributes are objects, too, as in the Smalltalk environment.

Iterative prototyping is a critical component of OT development. Throughout the project lifecycle, we continue to work with objects and their relationships; what changes is the level of detail. As each phase looks at objects from a somewhat different perspective, going though each step of a project lifecycle allows us to verify the correctness of our model.

As previously mentioned, the design should separate business functions (e.g., banking, insurance) from service functions (GUI, communications), allowing business objects, such as `Account` or `Client`, to be reused in any development environment, travel across the network, or be stored in a database. Separation is also essential to achieving a high degree of reusability. Further, the business model should reflect the business, not what is or is not possible in our development environment.

We need powerful visual tools to work with the hundreds of available classes. A good O-O diagraming tool and Smalltalk-like browsers are minimum requirements. We can also purchase classes from various vendors. Currently, most available software classes are for graphic user interfaces. Unfortunately, we cannot buy business-related classes such as `Account` or `Cheque` yet.

Coexistence

As OT is an evolution rather than a revolution, we need to manage the coexistence of new software developed using OT with existing software. We can use interface software, commonly called *wrappers*, to facilitate interactions between object-oriented and structured code. Wrappers convert (or wrap) code written with traditional methods into object-oriented code, allowing one to interface with legacy code in the OT environment.

Naturally, such an approach is a compromise. It has a number of constraints we will have to accept for quite some time. The main one is a conflict between two different information models: structured, with process and data separated vs. object oriented with both together. Also, it is difficult, if not impossible, to build a real ODBMS out of an RDBMS as the relational view of information stored may still be visible to the developer.

Today, we are already seeing some wrappers, e.g., for existing RDMBS products, that make it possible to use an RDBMS in Smalltalk or C++ environments.

Costs

The initial costs of the parts approach are higher than the craft approach because an OT foundation needs to be built. This includes training and education, new development tools, and new organization structures.

However, as we develop more reusable parts and better understand the technology, the development costs will drop below what we are paying today for the structured approach. The costs will decrease, however, only when we really change the paradigm and begin producing reusable software parts, readily available throughout the corporation. In the meantime, the cost of the traditional approach will keep increasing with the increasing complexities we have to handle.

The use of O-O development tools alone will not guarantee costs savings. If we remain software craftsmen, we will pay the additional costs of new tools without getting any significant payback from OT. We have to change the development process—parts we build have to be reused throughout the corporation—to ensure decreased costs.

SUMMARY

OT brings attractive benefits to software development such as: software quality, productivity, and flexibility at lower cost and shorter time to market. It greatly enhances communication among end users, analysts, designers, and programmers as all of them talk about nearly the same objects. These benefits, however, will be realized only after we change the way we develop software, i.e., move from craft to parts. Thus, the benefits will not be realized on our initial OT undertaking but rather some time in the future, depending on the speed of the transition. The change of tools is of secondary importance.

A number of companies are beginning to apply object technology: researching and evaluating it and building their first prototypes. Royal Bank of Canada has taken its first significant steps in exploiting this promising opportunity. We started by developing our expertise. Then, we began researching and evaluating the changes OT requires: new development methodologies, project lifecycles, class libraries, and tools. We completed a few OT prototypes from which we gained some banking parts that will continue to evolve with subsequent banking projects. ≡≡

Eugene Kulinek is currently working for Royal Bank of Canada as a Technical Systems Analyst. He is responsible for the introduction and evaluation of object technology at the company. He can be reached by phone at 416.348.5422 or by fax at 416.348.5460.

This suggests the natural locale for these services is a layer of the environment that knows nothing about objects. It also suggests the layers that do know about objects could package these object-unaware services so they are delivered in an object-aware manner. The natural consequence is a crisply defined layer that provides the OMS services in an object-model-independent manner.

This takes care of the diversity of object models that come from various vendors, programming languages, and technical traditions. However, this has only partly solved our dilemma between diversity and commonality! We are still bound to a specific set of visibility services. This set is usually selected by the vendor. For example, a DBMS vendor typically selects a single transaction-management service, versioning service, and so forth. Such rigid choices hardly take diversity into account....

We have to push the generalization one step further. We must provide functionality that can be turned into many variations of each kind of service, each variant dealing with the needs of a specific application domain, technical culture, or vendor's tradition. Instead of looking for the one-size-fits-all service, we should look for one common way to express any variation of this service. This leads to the idea of *metaservices* functionality that can be used to implement specific services.

A typical example is that of transaction management. Application domains vary considerably in their requirements for a transaction-management service. Online transaction processing (OLTP) applications typically need transactions that are short in duration, atomic in their actions on the data, and serializable in nature. Computer-aided design (CAD) applications typically need somewhat longer-term transactions that can be nested. Cooperative work (CW) applications may also need long-term transactions that are not serializable and can be arbitrarily shared among cooperating participants. Any attempt to provide a single, unifying transaction service to all of these applications results in overhead and underfunction. All get too much, and (paradoxically) all get too little.

The solution is that the lowest layer in our OMS must provide metaservices applicable to the most general data model we can imagine. Such a software layer would provide a common basis for a diverse spectrum of object management systems.

Thus, instead of providing one single, all-unifying transaction-management service this layer must provide primitives to express any such transaction models. The metaservices would be packaged by a layer above for export to particular application domains. Consequently, an environment for OLTP application would provide exactly OLTP transactions and likewise for CAD and CW applications. And yet, all domain-targeted environments would be able to seamlessly interoperate and cooperate due to the common lower layer they all use as a foundation.

THE DATA-STORING DOMAIN

Figure 1 shows the layered architecture along the data-storing dimension—the path between objects in an application's memory and objects in a secondary store.

The bottommost layer is that of the physical store. It is usually implemented in hardware but it may have software components. It presents to the layer above a very simple read/write/seek interface. The items of discourse are sectors containing uninterpreted bits.

The next layer up presents an upper interface operating on assembly-language-level untyped data. It provides both access and visibility services on bit-and-reference uninterpreted data. The items of discourse are data items made out of bits and references into other data items, as well as primitives controlling the visibility of such data items by any agent.

The next layer up presents an upper interface operating on typed (3GL-level) data. This layer introduces general-purpose typing semantics. This is analogous to the declarative aspects of a programming language. This layer also implements concrete transactions, versioning, sharing, licensing, configuration management, access control, and security models out of the primitives provided by the layer below. This is analogous to implementing the scoping and naming aspects of a programming language with persistence.

The next layer up presents an upper interface operating on objects. This layer introduces general-purpose object notions such as abstract data types, classes, inheritance, delegation, etc. It implements them out of the typed data primitives provided by the layer below. This layering is analogous to the object model (e.g., encapsulation, messaging, etc.) and class model (e.g., inheritance, dispatching, etc.) of an object-oriented language and generating code in a classic programming language.

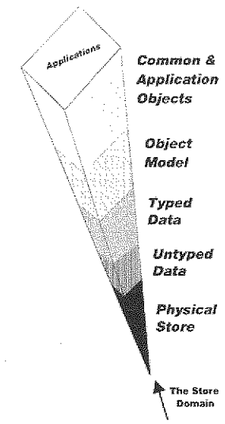


Figure 1.

ODBMS

Providing commonality while supporting diversity

by Sergiu S. Simmel

Like every other genre of environment software, object management systems (OMSs) are caught between a rock and a hard place.

They stand between the forces pulling them toward semantic (in content) and syntactic (in form) *diversity*, while their *raison d'etre* is to capture *commonality* and deliver it to many applications.

Diversity arises because each application makes different functional demands on the supporting environment. Object management systems require commonality to fulfill the promise of increased reuse under which object-based technologies are sold.

There is an obvious tension between the forces of diversity and commonality. On one hand, we must support each. On the other hand, we must support both.

The quality, endurance, and ultimate usefulness of an object management system (OMS) depends on how it resolves this tension. The OMS must allow for diversity, both in space (e.g., support for several paradigms or models) and time (e.g., ability to change when new applications come along). At the same time, the OMS must capture as much commonality as possible. The more essential common features it captures, the more useful it is, because the less the application needs to do.

Thus, the question remains: how can an OMS do enough but not too much?

Sergiu S. Simmel is a software author and developer in Arlington, MA. He has coauthored Kala, the persistent data server by Penobscot Research Center, Inc. He is a Software Architect for Oberon Software, Inc. in Cambridge, MA, and is working on a book about Kala to be published by Addison-Wesley.

THESIS

There are no easy answers. However, there is a general-purpose problem-solving technique that we can successfully apply: divide the problem into component subproblems and try to solve these many, but smaller, ones. This translates into what we have known for years as the *layered approach* to software architectures.

To make the OMS useful, we need to properly layer the object management architecture so it provides the functionality that common applications need while not being so rigid as to stifle opportunities for diversity. Such an architecture consists of a few well-defined, carefully crafted, and crisply distinguished layers with very sharp interfaces between them.

In a crisply layered architecture, we can migrate the common functionality towards the lowest few layers. These become the most stable and reusable layers in the architecture. The sharply defined interfaces insulate them from any change above. At the same time, the functionality subject to more diversity migrates towards the uppermost layers. These become the most variant and prone to replacement or change, least reusable, but most application-dependent layers of the architecture.

Experience proves crisp layering is much, much easier said than done. We all start with good intentions of clear principles and cleanly drawn onion structures but often end up with systems that look much like our onion after having been run over by a truck!

There is hope, however. Newer technologies have proven to be better able to carry through this ideal of clean separation. Lets take a look at one such example.

What is common about persistent objects?

An environment that manages objects provides two main categories of services. The first is managing objects themselves: their persistent states, relationships among them, actions taken upon them, etc. (i.e., the object model). The second category is managing the object's visibility to the accessing user: who can see an object and when. Visibility management includes managing transactions, versions, access security, concurrency, and licensing control.

Both object state and object visibility are services that retain much of their meaning when divorced of any specific object model.

INTERNATIONAL UPDATE

Object-oriented technology in Japan

by Daniel Levin

Japan is a remote and unknown place for most of us. Many US software companies do business with Japan, but few non-Japanese have a clear idea of business in Japan.

In this article, I will attempt to give an overview of what is going on in Japan relevant to object technology (OT) so you can identify concrete opportunities for yourself: a market for your software, ideas on how to develop an O-O application, and a warning about what your Japanese competition is up to.

I would probably get more attention by announcing Japan's imminent domination of the software industry than by telling the truth: Japan is not a leader in software development. The Japanese computer industry, with the help of the Ministry of International Trade and Industry (MITI), has made several unsuccessful attempts to leapfrog the international software technology state of the art with software factories, the Sigma project, and the 5th-generation project. The fact that almost no Japanese software is exported demonstrates that those efforts haven't made Japan a world leader in software development. OT is no exception: no O-O language or application has been exported from Japan. You don't need insider knowledge about Japanese software technology to judge the non-competitiveness of it any more than you need to visit a Japanese automobile factory to judge the competitiveness of their cars. But in both cases, you won't be surprised with the export results if you visit the production site. Japan may not be the leader in OT but in the rest of this article I will show why it is nevertheless a place of much interest for the OT community.

When they hear the word "Japan," most people think "exporter." But for software, Japan is definitely an *importer*, especially of the most basic packages like database management systems (DBMSs), operating systems (OSs), and network management software. Japan is the world's second largest software market and worth any software vendor's attention. My company obtained 30% of its revenue from Japan in 1990. Concerning the practical issues of selling your software in Japan, I suggest that you read the AEA's handbook.[1]

Although Japan is not a leader in OT, I agree with the average doomsayer that it is necessary to keep an eye on Japan to check if some company is going to copy your idea, improve on it, and flood your market. Japan did that for hardware, and has been attempting to do the same for software. A look at some representative Japanese OT projects will give you an idea of both the specific interests and the state of deployment of OT in Japan.

- Fujitsu (Jasmine), Hitachi (Mandrill), and NEC (Odin) have developed prototype object database management systems (ODBMSs). For previous generations of DBMSs, they provided proprietary DBMSs on proprietary hardware. Now that they are selling more and more open systems they are evaluating when to provide ODBMSs to their customers and whether to develop proprietary ODBMSs.
- A hardware vendor has developed a prototype office information system that stores, links, and retrieves various types of office information: fax, email, voice, word processor text, etc.
- ATR, a research organization owned half by Nippon Telephone and Telegraph (NTT) and half by the government, has developed a prototype CASE tool that stores source code in an ODBMS and displays the structure of it as a graph, thereby aiding modularization and reuse of software modules.
- Sony is developing an interesting O-O research project: MUSE, an O-O operating system that enables all users to access all computer resources regardless of location.

Daniel Levin is the Regional Director of the Japan Office for Servio Corporation.

- Ship and Ocean Foundation is working on a prototype of CIM for shipbuilding. Currently, design information is scattered into numerous incompatible proprietary file formats. Whenever a change to ship design is required, the change ripples through the various systems wreaking havoc as it goes. Ship and Ocean Foundation hopes that putting all information into one ODBMS would cut the total production time-span in half.

> ***...if your company makes anything that by any stretch of the imagination can directly or indirectly incorporate OT, you'd better not rest on your technological laurels.***

The various projects above will not make many of you worried about Japanese competition, but there are a few fields in which you must reckon with it. For example, multimedia is receiving much attention in Japan. I am responsible for sales of GemStone, Servio's ODBMS, in Japan. Over half our customers and prospects are interested in OT to handle multimedia applications. A visit to the Sony Computer Fair illustrates the state of multimedia in Japan. As a corporate strategy, Sony is trying to create a synergistic whole of all its activities: video cameras, CD players, workstations, palmtops and laptops, movies, music, cartoon characters, etc. Other Japanese hardware vendors have similar, although less broad, multimedia strategies. Sony has developed its own workstation and PC and organized the Sony Computer Fair to let software vendors exhibit their applications. Over one-third of the applications exhibited are multimedia. The most typical demo program, whether it is for a CD ROM image database or a GUI development tool, is a personnel database. What is so important about putting photos into a personnel database? Can't you do without them? That's the point Japanese vendors and customers are making: most data that companies manage is multimedia. For example, the typical personnel department has employees in a database. In a filing cabinet, they have a file for each employee with a picture, handwritten performance evaluations, a photocopy of a resume, etc. The computer is handling one-tenth of the information, the other nine-tenths is being handled by manual updates, fax transmission, photocopy distribution, phone inquiries, etc. The Japanese market contradicts pessimists who say multimedia is a technology but not a market. Japanese hardware vendors have identified the market for multimedia: managing the kinds of information found in any office: voice, pictures, diagrams, spreadsheets, texts, etc. I don't think that the 6th-generation project (successor to the 5th-generation project) has any chance of making a significant impact on the quality of Japanese IT, but the efforts devoted to multimedia could have a major impact on the competitive position of both Japanese hardware vendors and their Japanese customers.

No matter how well you master OT, if your company makes anything that by any stretch of the imagination can directly or indirectly incorporate OT, you'd better not rest on your technological laurels. Consider "fuzzy logic" as an example of a not-particularly-Japanese technology the Japanese industry turned into a competitive advantage. Fuzzy logic is a field of AI. Japanese electric appliance manufacturers have incorporated it into washing machines, ovens, and other electric appliances to enable them to make their own decisions about how to do their job. "Fuzzy" for Japanese electric appliances has a status similar to "no cholesterol" or "with bran" for food in the USA: you can't sell your product if it doesn't have "Fuzzy" written on it. Using fuzzy logic both to add intelligence to appliances and as a sales point was a great strategy, and I expect to see the Japanese be the first to label automobiles and fax machines "object-oriented."

How about *benefiting* from the results of Japanese research, for a change? The Sigma project is an example of what not to do. This ambitious project, implemented by a government-sponsored industry consortium, aimed to standardize all Japan on a common software-engineering platform including a standard hardware platform, OS, languages, and CASE tools. Standardization was to enable reuse of software "parts," and the combination of reuse and CASE tools was projected to yield a 10x increase in productivity. Last year a cover story in NIKKEI COMPUTER (the Japanese equivalent and corespondent of DATAMATION) was titled "The failure of Sigma." The main reasons pointed out for Sigma's failure were:

- Premature standards. By the time Sigma was completed (1990), all the standards were obsolete, outweighing the alleged benefits of standardization.
- Insufficient technology for reuse. Calling the same old pieces of C or COBOL code "software parts" didn't make them reusable.
- Lack of commitment. The typical Japanese government's strategy of making competitors cooperate to develop new technology didn't work because the technology was not precompetitive.

There are also some lessons to be learned from Japan regarding what you *should* do. Of course, I suggest you try to emulate Japan's successes rather than failures. Japan's impressive success in product design, although not directly related to software development, should inspire all software developers.

Many people in the field of software engineering, particularly in the field of OT, consider "industrialization" of software development to be a desirable goal. Don't waste your time studying conveyer belts and tire molds unless your job is to produce multiple copies of identical software. The crisis is in software development, not software reproduction. Development is a design process, so inspiration should come from studying the way automobiles and buildings are designed. Japanese companies are world-class product designers. Some of their techniques are relevant to O-O software development.

Rule 1 is to produce a design that satisfies all its users. To ensure each department has other departments' needs in mind, Japanese employees are rotated through various departments of their companies. Also, people from all stages of the production process are included in the design team from the start. For software development, the lesson is that superior coordination of various departments yields better results than superior professionalism in each department.

Rule 2 is that trying to get a design right the first time is not the key to success: iterative design is. I have no idea who invented the telephone answering machine, but the first time I went to Japan, in 1987, I saw that Japanese manufacturers were selling answering machines that were smaller, better looking, and easier to use than Western ones. Western answering machines were an implementation of the first iteration of design: a tape recorder attached to a phone. Thus, they were machine-centric: to listen to your messages you pushed "rewind" to rewind the tape, "play" to play back the messages, then "stop," and then "rewind." Meanwhile, Japanese answering machines were already several iterations beyond the first design and had already become user-centric: you pushed "play" to hear your messages.

Rule 3 is to reuse designs. The Japanese know a lot about reuse. One example is architectural firms. They have people traveling around Japan and the world taking pictures of buildings. They manage that competitive data in addition to data on their own productions. The most advanced companies are looking into using ODBMSs to manage their databases. The point is: reuse is not cheap. It requires a big investment in managing the reusable designs. Japanese companies also illustrate what champions of OT have been saying for years: you have to reward employees for putting in the extra effort to make a design reusable, which makes the company more productive, rather than rewarding developers for botching their job as quickly and selfishly as possible. One more skill required for reuse that the Japanese have mastered is humility. Japanese tend to assume nobody is perfect and that the design they are reusing is probably better than what they would have created themselves. Westerners usually assume the opposite, a major obstacle to reuse.

To conclude, let's summarize the main things you can do. You can sell software to Japan. You should watch out for the way they incorporate technology into products like the ones your company sells. And, you can learn about how to produce designs from Japan. ≡≡

Reference

1. SOFT LANDING IN JAPAN, American Electronics Association, Japan Office.

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: Robert Shelton, 1850 Union Street, Suite 1584, San Francisco, CA 94123; fax: (415) 928-3036.

Sept. 21–25, 1992 **C++ By Design** New York, NY Contact: 212.274.9135	Sept. 30–Oct.2, 1992 **CASE World** Boston, MA Contact: 508.470.3880	October 13–14 **Yourdon on Object-Oriented Analysis & Design** Chicago, IL Contact: 508.470.3880	October 18–22, 1992 **OOPSLA '92** Vancouver, BC, Canada Contact: 407.628.3602	Nov. 16–20, 1992 **C++ World** Meadowlands Hilton, NJ Contact: 212.274.9135

hotline on
OBJECT-ORIENTED
technology

Back issues

All back issues of the HOTLINE are available. Please call 212.274.0640 for details.

Vol.3, No.10/August '92 ▪ Object technology: toward software manufacturing ▪ Return on investment: software assets and the CRC technique ▪ Object-oriented technology in Japan ▪ Providing commonality while supporting diversity

Vol.3, No.9/July '92 ▪ OOD: Research or ready ▪ Enterprise modeling: an object approach ▪ OMG's 18–24 month view ▪ Design for object-oriented applications: a CASE for wishful thinking...

Vol.3, No.8/June '92 ▪ Business in the Information Age ▪ From data modeling to object modeling ▪ How frameworks enable application portability ▪ Interview with Vaughan Merlyn

Vol.3, No.6/April '92 ▪ Thinking the unthinkable: reducing the risk of failure ▪ Mitigating madness with method: first establish what you value ▪ Championing object technology for career success in the 1990s ▪ Objects and actions in end-user documentation

Vol.3, No.5/March '92 ▪ TA large-scale users' assessment of object orientation ▪ Report on the Object-Oriented COBOL Task Group ▪ Interview with K.C. Branscomb

Vol.3, No.4/February '92 ▪ The big prize: acceptance of O-O by the MIS community ▪ Retrospective: 1991—the year it all changed ▪ Making the transition to O-O technology ▪ Interview with Beatriz Infante

Vol.3, No. 3/January '92 ▪ Enterprise object modeling: knowing what we know ▪ Adopting objects: pitfalls ▪ Adoption rate of object technology: a survey of NSW industry

Vol.3, No. 2/December '91 ▪ Accepting object Technology ▪ Adopting objects: a path ▪ Incorporating graphical content into multimedia presentations

Vol.3, No. 1/November '91 ▪ Leading the U.S. semiconductor manufacturing industry toward an object-oriented technology standard ▪ Coping with complexity: OOPS and the economists' critique of central planning ▪ Choosing Object Technology: What's the object? ▪ OOP: the MISsing link

Vol.2, No. 12/October '91 ▪ A modest survey of OOD approaches ▪ What is a "certified" object programmer? ▪ Perspective: investing in objects today ▪ Object oriented in Melbourne, Australia ▪ The Object Management Group

Vol.2, No. 11/September '91 ▪ From applications to frameworks ▪ Report on the Object-Oriented COBOL Task Group ▪ Getting started with object technology: efffectively planning for change ▪ Object statistics on the way ▪ On objects and bullets

Vol.2, No. 10/August '91 ▪ Distributed object management: improving worker productivity ▪ Getting the best from objects: the experience of HP ▪ APPLICATIONS: EC employs object technology . . . ▪ CAPACITY PLANNING: Fiddling while ROMs burn

Vol.2, No. 9/July '91 ▪ Multimedia is everywhere! ▪ Developing an object technology prototype ▪ Object-oriented capacity planning ▪ How OOP has changed our developmental lifecycle ▪ Modularization of the computer system

Vol.2, No. 8/June '91 ▪ Domain of objects: the Object Request Broker ▪ Object-based approach to user documentation ▪ Report on the Object-Oriented COBOL Task Group ▪ Do we need object-oriented design metrics?

Vol.2, No.7/May '91 ▪ Hybrid object-oriented/functional decomposition for software engineering ▪ So, what makes object databases different? (Part 4) ▪ Using the generic application to solve similar domain problems ▪ Experiences using CLOS ▪ International Conference on Object-Oriented Technology, Singapore

Vol.2, No.6/Apr. '91 ▪ An artist's perspective of programming ▪ So, what makes object databases different? (Part 3) ▪ Moving from Pascal to C++, Part 3 ▪ Object projects: what can go wrong ▪ Reflections from LOOK-'91

SUBSCRIBE NOW TO THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY— DON'T MISS ANOTHER VALUE-PACKED ISSUE!

❑ *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)*
(outside US add $30 per year for air service)

Back issues @ $25 each *($27.50 foreign):*
Vol.2, Nos. _____
Vol.3, Nos. _____

❑ **Phone/fax order**
Call Subscriber Services at 212.274.0640
or fax this form to 212.274.0646

❑ **Bill me**

❑ **Check enclosed**
Make check payable to the HOTLINE and mail 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)

❑ **Credit card orders**
❑ MasterCard ❑ Visa ❑ AmEx

Card# Expiration Date

Signature

Name

Title

Company/Mail Stop

Street/Building#

City/Province

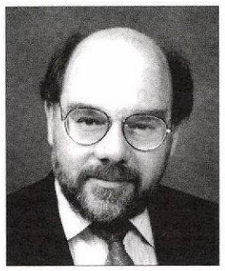
ST/Zip/Country

Telephone

hotline on
OBJECT-ORIENTED
technology

VOL. 3, NO. 11 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES SEPT. 1992

Developing strategic business systems using object technology



Dr. Jerrold M. Grochow

Over the past two years, the thundering waves of object orientedness have become deafening. We are bombarded with articles, speeches, and pronouncements on object technology as the savior of our industry. We are inundated with advertisements and product literature on the latest books, languages, and tools to make object design and development painless. We are exhorted to "think like an object, act like an object, be an object," as though somehow this will save the world.

Unfortunately, a great deal of this cacophony is mere rehashing for those of us in the information systems (IS) part of the industry (a.k.a. business systems, management information systems, etc.). We have heard it all before: "Use structured techniques and your systems will be maintainable"; "Use 4GLs and you will eliminate the application backlog"; "Use CASE tools and you will improve productivity 50–100%"; "Object technology will deliver on all the promises that other technologies didn't." Yes, and "the check is in the mail."

If object technology is really going to help us, we need to address the "real" problems in designing and developing strategic business systems. When trying to figure out how to extend the life of existing systems of more than a million lines of COBOL code (mostly undocumented) running as a standalone system on a mainframe computer, the issue is not whether C++ is object oriented enough or which dialect of Smalltalk to use. The real issues in object technology are:

- will it scale?
- can it be managed?
- will it perform?

The question is whether object technology will provide value to the IS manager in the form of robust tools and methods for creating heavy-duty systems. The IS manager is searching for new methods that must apply to the large-scale, high-volume situations faced everyday in the world of business systems. With hundreds (or thousands) of online users making split-second decisions regarding millions of dollars, the brokerage house IS manager can't explain that a strange lull in system response is due to the environment "garbage collecting." When nightly production schedules can't be missed without risking a Federal audit, the bank IS manager can't say that the new object-oriented system "doesn't handle batch processing so well." With database sizes measured

continued on page 10

IN THIS ISSUE

FROM THE EDITOR

During July, I had the pleasure of chairing the London ObjectExpo Executive Briefing, a panel discussion organized to familiarize senior and mid-level managers with object technology. The majority of attendees managed Information Technology (IT) groups and divisions in the UK, from finance to basic industry, government to health care. Our panelists were leaders in the object technology vendor and user community.

Over 50 managers participated. A straw poll taken at the start of our session revealed that three had been involved with object technology for more than one year, while just under a dozen had less than a year's experience. Of the remainder, half had a reading familiarity with the subject area and the other half were undertaking their first exploration.

Their questions were as revealing as our panelists' responses: How does this technology scale in the large? Can it be used for fault-tolerant or real-time systems? Can objects be used in an online transaction processing application? One of the conference's most enlightening speakers, a panelist and a regular HOTLINE author, Mr. Norman Plant, Chairman of the Object Interest Group, challenged the technology's current suitability in large-scale core applications—an area in which some of our other panelists had surprisingly limited hands-on experience. His experience-based concerns were echoed by a question about handling the rapid-application development requirements of the financial community, wherein a trading house might require applications to develop and deliver a new financial instrument or to take advantage of an opportunity with a life-window of only a week or two.

These are the real day-to-day concerns raised by this month's feature author, **Dr. Jerrold Grochow** of American Management Systems. He points out that much of the hoopla over objects is rehash for the IT manager, and that the fundamental questions about objects, as with any new technology, are about scalability, manageability and performance. Real life for IT managers running major projects is not radio buttons and pastels patinas—it is budget, schedule, and satisfied customers. Dr. Grochow would agree with the panelists' conclusion: object technology is profoundly useful in these arenas, but only after overcoming the barriers to entry. Some of these—lifecycle and methods, training and education, tools, and management control—are addressed by our other authors this month. But Dr. Grochow raises several critical needs that are not receiving wide attention as yet: a design repository that works, methods for estimating development time and cost, and methods for estimating operating costs. I agree that we have not adequately met these needs in the traditional software development domain but, as Mr. Grochow points out, managers expect much more than the platitudes of the sales pitch. Just saying that object development will be cheaper and faster is simply not good enough—especially for the managers who have to bring their organizations' first object technology project to success. Remember, success is results against expectations, not just results! If the sponsoring manager expects the first object-oriented project to cost less and be completed more quickly, yet the result conforms with industry experience (taking two to three times as long as it would using conventional development methods and programming technology), then the project has failed in the eyes of the beholder!

Reinforcing our author's point of view, Mr. Gordon Eubanks, President and founder of Symantec whose perspective on object technology is based on his employees' adoption experiences, reminded us at the Executive Briefing that much of what we place under the umbrella of object technology is not new or revolutionary. It is rather common sense, often the same common sense that our industry has fallen short of in the past! Furthermore, what is tough about object technology is the adoption/training process, not the concepts of the technology itself. Dr. Adele Goldberg, Chairman and founder of ParcPlace Systems, and also a Briefing panelist, reinforced the importance of managing expectations through the adoption process with statistics from 38 object technology projects. Dr. Goldberg's and other studies are beginning to address the hard questions that information technology managers must ask of object technology and its vendors, but I expect most of us would agree, despite our enthusiasm for this technology, that the answers are far from solidified as yet!

Mr. Sam Adams of Knowledge Systems Corporation brings us the second in his four-part series on the class-responsibility-collaborator method. CRC takes an interactive approach to modeling the behavior of a system and its external compatriots. As such, it will likely be the methodological counterpoint to the data-structure/behavioral approaches that are growing out of entity-relationship (ER) modeling.

This article addresses one part of the problem Dr. Grochow identifies: method and lifecycle sufficiently comprehensive to be applied at the enterprise level, capable of handling complex industrial-grade systems. Certainly CRC provides a uniform model from top (enterprise) to bottom (class hierarchy design). The challenge, however, remains to identify and define the transformations that take place through this range. As work with the Zachman Framework has led us to understand, there are multiple views on any system. Mr. John Zachman, formerly a senior consultant for IBM Corporation, has refined these views into layers that are each meaningful and distinct. The layers are not decompositions of one another, nor is the difference simply at the level of detail and exactitude. Each layer represents a different view of the same system: the ballpark view, or strategic vision, as seen by the CEO; the business tactical view; the information technology view; etc. Thus a CRC model of the ballpark view (enterprise model) would be expected to differ from the business view (business conceptual model),

toward quality. It is not sufficient to test completed applications for defects, remove the ones we discover, and ship the rest to customers. We must instill quality at every stage of development. And we must be satisfied with nothing less than zero defects. With conventional software, this is almost certainly an unrealistic goal. But object technology presents new opportunities for improving the lamentable state of software quality. . . .

A quality-first program for object technology, David A. Taylor, OBJECT MAGAZINE 7-8/92

. . . Robert E. Lee said "Plan no more than necessary." His ultimate defeat was probably due more to the implementation of this philosophy than its validity. The problem in development, again, as in war, is how to know when to stop planning and start moving. The answer is never stop planning but never let planning prevent progress. The best methods today facilitate iterative development. Use one with object-oriented techniques for the appropriate tasks to get the most powerful and compete approach available.

Planning, lookahead,and spiraling into control, Adrian Bowles, OBJECT MAGAZINE 7-8/92

. . . [Steve Jobs:] Software development today is collapsing under its own weight. It has gotten too complex for the current way of development. Object-oriented software allows you to hide and manage great complexity. That means you can build much more complex software. That's the biggest benefit. The second biggest benefit is you can build a lot faster. The third biggest benefit is it is much more reliable. Then you get into reusability issues. If a company is writing ten applications and each one posts some debit or credit in a general accounting system, ten different groups of ten people at ten different times will write different pieces of applications to do that. Let's say that you decide to go into Europe and you have to support multiple currencies. You've got ten things to change. If they all use the same posting object, with some subclass changes but basically the same object, then you can change it in one place and it will dynamically link into all applications automatically. Those benefits are just beginning to be discovered but they are going to be huge. . . .

The Next step and beyond: a conversation with Steve Jobs, UNIFORUM MONTHLY, 5/92

. . . My experience has been that the shops that were the most successful with structured techniques are also the most successful with object-oriented techniques. This correlation isn't strange. Structured techniques are successful only in shops that have direction and foresight, good team communication, discipline, and management commitment to up-front analysis and design. Object-oriented techniques are, in many ways, like structured techniques—only more so. Success in deriving useful classes, sound class hierarchies, and other object-oriented structures requires a degree of software-engineering discipline that surprises even some experienced software engineers. Shops that have failed with structured techniques should examine carefully the reasons for their failure—not merely declare that "structured techniques don't work"—before adopting any other set of techniques. This is especially important if those techniques are object-oriented. . . .

Object orientation: the importance of being earnest, Meilir Page-Jones, OBJECT MAGAZINE 7-8/92

. . . OOP, as presented by its proponents, represents a return to basics. The emphasis is on the realistic modeling of the user's world, and a systematic approach to design and programming. In addition, OOP languages have features that facilitate adjustments resulting from mistakes and the evolution of the business. Object-oriented systems will not be written by novice developers; in fact, overall skill requirements are likely to be greater . . . A problem is that object-oriented languages are a revolutionary, not an evolutionary step from earlier languages . . . Their concepts are new, and there will be large learning curves for any enterprise taking up the approach. This will be costly. In addition, the enterprise will benefit only in the long run over a number of projects, and only if the entire enterprise commits to the new approach . . . The object-oriented variations of the standard 4GL are not likely to cause any immediate change in the current use of 4GLs. However, in the next three years, the trend to object-oriented languages will change the look of current 4GLs and eventually lead to their more widespread use.

No gain without pain?, Gerald Adams, SYSTEM DEVELOPMENT, 5/92

. . . The ability to choose an access method enables new users of object technology to choose an access method with which they are familiar. Many of these new users, living for years within traditional data management and design concepts, find object-oriented concepts divergent and esoteric The use of a familiar access method and database design shortens the time to implementation. As proficiency and understanding of object-oriented concepts improve, the other access methods can be learned and later applied to enhance a database design. . . .

Access methods, Grant Colley, OBJECT MAGAZINE 7-8/92

way. He then went back to Luigi and was able to report that all his programmers were now using C++ (developed by the very guy who was sitting under the tree in BUSINESS WEEK). Since C++ is an object-oriented language, we must be an object-oriented shop. Everyone was happy. The programmers barely noticed a difference. Except for a few brave experimentalists, none of them changed coding habits. Nosbert had saved face without having to do anything significant. Luigi could hold his head high now that [his] company was again basking in the white heat of modern technology. I believe that the technical term for all of this is "declaring victory." . . .

Object orientation: the importance of being earnest, Meilir Page-Jones, **OBJECT MAGAZINE**, 7-8/92

. . . At a recent IBM briefing, Jim Cannavino also seemed to downplay Taligent's impact on IBM's current PC operating system strategies. On the eve of the shipment of OS/2 2.0, Cannavino said he saw OS/2 enduring throughout the decade; he even saw it moving to other platforms (RISC, in this context). This caught us by surprise, as we thought Taligent was specifically for the RISC platform. Both Apple and IBM mentioned that they would use the research from Taligent to enhance their current products, including the addition of object-oriented technology, both to prolong product life and, presumably, to offer a path (eventually!) to Taligent itself. What's going on here? Has Taligent been orphaned at birth, abandoned at the start by the very companies that paid for its development and whom we thought cared about its future? It is much more likely that we're seeing naturally cautious senior-management types exercising prudence about a product whose potential impact and revenue stream is still far, far away. . . This product hasn't been orphaned at birth; the parents just don't want to claim this child prematurely. They're waiting for the intelligence test results; they want to see how many baby beauty contests it's going to win. . . .

Taligent orphaned at Birth?, Amy Wohl, **COMPUTER SHOPPER**, 6/92

. . . According to projections from computer industry market research firm International Data Corp., the OODB [Object-oriented database] market will double each year, over the next five years. . . Already, OODBs have established a presence in mechanical CAD (computer-aided design) and multimedia environments where users are dealing with complex information that needs to be showed and retrieved quickly. One of the key advantages to OODBs is that they can more easily handle complex data like graphics, whole documents, multimedia applications, e-mail and voice mail. OODBs also handle high levels of abstraction and can represent, store and retrieve all the data pertaining to an entire company, for example, as a single object. With RDBMSs, complex data has to be broken down and filled into rows and columns, like a big spreadsheet and complex queries hurt performance. . . Although OODBs offer advantages for complex data types, there are areas, such as online transaction processing applications (OLTPs), and retrieving content based on single values, where RDBMSs are faster. Another hurdle for object technology is the heavy investment corporations have in relational technology. Customers want to have a clearly mapped migration path to object technology before they jump in with both feet, says Robert Marcus, head of the newly formed Corporate Facilitators of Object-Oriented Technology (C-FOOT). . . Is the hybrid approach simply a stopgap measure on the way to full object databases, or can we expect this approach to be the dominant database model of the future? According to Oracle's David Beech, senior product manager of Object Systems, the evolution will be for relational systems to make more use of object technology and graphical interfaces: "I see a convergence of the two fields. One is just more general than the other." . . .

The next wave: object oriented databases, Gordon Arnaut, **INFO CANADA**, 5/92

. . . According to a spokeswoman, interest in object technology is reflected in a telephone survey Object Expo sponsor SIGS Publications made last month. The survey showed that 75 percent of the Fortune 100 industrial companies are using some object-oriented technology. Half of the remaining quarter said they'd consider it next year.

Object Expo Tools, Dan Richman, **OPEN SYSTEMS TODAY**, 6/22/92

. . . The hottest new development technology is object-oriented programming. It's so hot that vendors are proclaiming new versions of old software to be "object-oriented" about as fast as they can yell the words. It's so hot that you might be tempted to dismiss it as just so much hot air. Don't. . .If you're developing software any other way two years from now, you're probably making a mistake.

Object-oriented programming is worth its cost, Mark L. Van Name & Bill Catchings, **PC WEEK**, 6/15/92

STRATEGIES

. . . Object technology is often compared with manufacturing because new applications are assembled out of existing components rather than crafted from scratch. In terms of testing, however, we are emulating an outmoded model. We leave testing until our products are completed, and we ship software that is virtually certain to contain serious defects. If we are to emulate the success of modern manufacturing, we must take the same attitude

NOTE TO OUR READERS:

To make it easier to save and protect your copies for back reference, the HOTLINE has been redesigned to fit into a standard three-hole punch looseleaf binder.

Customized HOTLINE binders hold two volume years and can be purchased for $15 (including shipping and handling) by calling 212.274.0640.

which in turn would differ from the information systems view (class hierarchy). This powerful and essential yet subtle concept remains to be fully appreciated in the information modeling community. Methods such as CRC are being evolved into enterprise tools from the bottom up, and will have to address these perspectives as we come to clearly define them.

Our author **Dr. David Bellin**, an educator and trainer, takes a hard look at training. Object training is not simply a one-week class in an object-oriented programming language. A class in C++ for the programmers will not bring substantial benefits to the organization. We are reminded of Dr. David Taylor's books, which speak of the need for training and education. Ms. Elizabeth Sevean, founder of the New York Object Users Group, has picked up this theme by beginning to develop a curriculum for object technology. Our industry is on the tip of an iceberg. We know that mentoring is critical to successful training, as in the case of structured methods, ER modeling, and CASE and conventional programming. Organizations lean on the macho programmer fantasy—you know, "real programmers don't need. . ."—as a short-sighted excuse to avoid investing in their own professionals. Right! And the check is in the mail. . . .

Mr. Robert Howard of Rock Solid Software takes on the issue of process management through Total Quality Management (TQM). As you can see from reading Mr. Adams' series, the process by which systems are developed largely determines the quality of the results. This is basic TQM. Mr. Howard addresses the tactical question: How do we bring these management techniques to bear at the programmer/developer level, in our project teams and in our organizations? In last month's issue, Mr. Daniel Levin of Servio Corporation Japan Office discussed TQM as one of the US and EC's great vulnerabilities. We created this powerful technology yet our ability to master management lags behind those who gladly import and use our technology to their competitive advantage. Anyone who says Lee Iaccoca of Chrysler Corporation cries foul over Japanese competition while promoting the Dodge Stelt (a car completely manufactured in Japan by Mitsubishi) should appreciate that it's high time to stop grousing and start learning from one of our best customers and competitors. TQM is a pivotal management tool; just talking about it will not make us competitive. TQM is about action, not talk.

We have yet to address tools for distributed deployment or configuration management, integration of a transaction management layer such as AT&T Tuzedo or the impact of a CORBA-compliant layer. We need to look at development tools or the suitability of object database management systems for large organizations like British Airways or Bank of America. There is much ground to cover on issues raised by authors like Mr. Plant and Mr. Grochow.

We are beginning—perhaps just beginning—to address the issues that will make this technology approachable and useful to the IT manager. The issue is not whether IT managers are interested in object technology, but how they will make it work. Vendors and early adopters share the burden of communicating the how-and-why of their successes and failures so that interested managers can make a reasonable case for using this technology in their business. = =

hotline on OBJECT-ORIENTED technology

Robert Shelton, Editor

SIGS ADVISORY BOARD
Tom Atwood, Object Design
Grady Booch, Rational
George Bosworth, Digitalk
Brad Cox, Information Age Consulting
Chuck Duff, The Whitewater Group
Adele Goldberg, ParcPlace Systems
R. Jordan Kreindler, General Electric
Meilir Page-Jones, Wayland Systems
Tom Love, OrgWare, Inc.
Bertrand Meyer, Interactive Software Engineering
Sesha Pratap, CenterLine Software
P. Michael Seashols, Versant Object Technology
Bjarne Stroustrup, AT&T Bell Labs
Dave Thomas, Object Technology International

HOTLINE EDITORIAL BOARD
Jim Anderson, Digitalk, Inc.
Larry Constantine, Consultant
Mary E.S. Loomis, Versant Object Technology
Reed Phillips, Knowledge Systems Corp.
Trygve Reenskaug, Taskon A/S
Zack Urlocker, Borland International
Steven Weiss, Wayland Systems

SIGS Publications, Inc.
Richard P. Friedman, Founder & Group Publisher

ART/PRODUCTION
Kristina Joukhadar, Managing Editor
Susan Culligan, Pilgrim Road, Ltd., Creative Direction
Elizabeth A. Upp, Production Editor
Jennifer Englander, Art/Production Coordinator

CIRCULATION
Diane Badway, Circulation Business Manager
Ken Mercado, Fulfillment Manager
Vicki Monck, Circulation Assistant
John Schreiber, Circulation Assistant

MARKETING
Lorna Lyle, Promotions Manager—Conferences
Sarah Hamilton, Promotions Manager—Publications
Caren Polner, Promotions Graphic Artist

Administration
David Chatterpaul, Bookkeeper
Ossama Timoum, Business Manager
Claire Johnston, Conference Manager
Cindy Roppel, Conference Coordinator
Helen Newling, Administrative Assistant

Margherita R. Monck, General Manager

Jane M. Grau, Contributing Editor

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012, (212)274-0640. Mailed First Class. Subscription rate — one year (12 issues) $249, Foreign and Canada $279. Single copy $25.

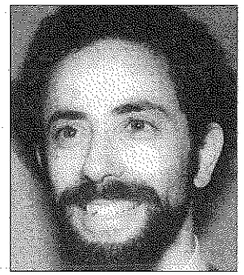
POSTMASTER: Send address changes & subscription orders to HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville, NJ 07834.

Submit editorial correspondence to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123 voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of HOTLINE ON OBJECT-ORIENTED TECHNOLOGY, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, THE X JOURNAL, C++ REPORT, THE SMALLTALK REPORT, and THE INTERNATIONAL OOP DIRECTORY.

Object training: harder than it looks



David Bellin, PhD

Object technology is today's most promising software development technology, offering great hope for overcoming some of the worst Achilles' heels of the software development community. The promise of substantial savings in development and maintenance costs seems near but, before these savings can be realized, a substantial training effort must be undertaken. This costs time, effort, and money. Most important, the training task must be approached systematically to produce the expected payback. A Fortune 500 training manager told me, "Object-oriented techniques have proven to be the most expensive and most difficult training project we have ever undertaken. It's costing us much more than we expected." This manager is not the only one with this experience. In fact, programmers of object-oriented (O-O) languages uniformly agree that the transition to a fully O-O approach is the most difficult learning task of their careers. Early training is one of the most crucial steps to adopting the technology successfully. Developing good training programs, timing them, and overcoming obstacles in order to implement them are all difficult tasks.

Object technology newcomers frequently underestimate the extent of their educational needs. An experienced structured programmer needs a minimum of six months of practice to become proficient in a pure O-O language. This was supported by Mark Lorenz of IBM in a recent Hotline article,[1] who stated that it could take up to a year for true object-oriented programming (OOP) proficiency to be reached. All the OOP experts I have spoken with have confirmed this estimate.

Clearly, this learning curve is a great deal longer than when structured languages were in vogue. Why this is so may be open to debate and discussion. Personally, I think conceptualizing systems in O-O terms is actually harder than doing so in process-oriented, data-transformation terms. Most disagree; they feel that objects are a more "natural" way of modelling the world, and blame the difficulty on differences in syntax and large class libraries with which the coder must become familiar. But whatever the reason, it is clear to all who have adopted object technology that programmers need substantial education.

> ***Most successful training programs combine formal and informal methods. Both are crucial to OOP.***

Most successful training programs combine formal and informal methods. Both are crucial to OOP. I have found it successful to initiate the process by providing time, manuals, and software for programmers to "play" with for several weeks. This play-exploratory period is followed by a formal seminar in OOP using the particular platform being adopted. After a one-week breather, it is a good idea to hire an outside consultant. This OOP expert is needed to provide the programming expertise that probably cannot be found in-house while the organization is still in training. After a month or so of active work with OOP, most development teams are able to start some internal education dissemination, in which the more skilled or rapid learners on the team can begin to help their peers. This has been the most successful approach. I have used it in my own consulting work, where we strive to make the clients comfortable with the new technology by the time we complete an extended visit on-site.

No successful data processing (DP) shop achieves its results solely from outside expertise, so a key decision is how to breed knowledge in-house. There are many approaches to this. The one I favor involves development of a leadership group, similar to the elite police "SWAT" teams whose training and personal standards are expected to provide an example and inspiration to the rest of the force. How this OOP SWAT team is handled has a significant imact on the results of your organization's OOP training. The SWAT team should consist of a core group of four to six programmers who are put to work on a small test project so that they may become thoroughly conversant in O-O technologies. This team then becomes the core for further technical education within the organization. In other words, they are groomed as the in-house "gurus."

Then there is the problem inherent to all gurus, from Timothy Leary to Pat Roberts, who may come to consider themselves virtually perfect and indispensable. This can lead to both extreme salary demands and inferior organizational performance. However, gurus also have a lot to offer: they know their stuff, and can be gratified by disseminating the word to others. My experience is that good managers can handle any potential difficulties and

ented programming is not really new at all. It can be traced back to 1967 and the work of Nygaard and Dahl in creating the Simula language. Its basic concepts are already over 20 years old. The time it has taken to win broad acceptance is very similar to the time it has taken the concept of "open systems" to become accepted. The long gestation period of these two very important ways of thinking about software may tell us something about how fast powerful new ideas are accepted by the computer marketplace, at least on the software side. . . .

Object-oriented methodology, OPEN SOFTWARE JOURNAL, Vol 5/issue 1 1992

STANDARDS

The Open Software Foundation, Inc. (OSF) said last week it has scrapped plans to use IBM's data engine in the initial release of its Distributed Management Environment (DME), saying the object-oriented software was too complicated to integrate with other technologies in the systems and net management framework. . . OSF said it will now rely on other components within the DME, including Tivoli technology, to fill the role planned for the data engine. However, the group has not ruled out implementing the IBM data engine in subsequent releases of DME. . . .

OSF scraps plan to use IBM data engine in its DME, Jim Duffy, NETWORK WORLD, 6/15/92

CLIENT/SERVER

. . . Converting a terminal-based application to a client/server paradigm using a GUI running on a PC misses the point, because such applications will not take full advantage of the horsepower that is available on the desk top, [Aaron Zornes of Meta Group] explained. "Clients should not have to wait for acknowledgement from a server. What is needed is an asynchronous message-based system where clients are dedicated to servers. With such an approach, you can optimize specific servers for certain types of applications," he said. "The goal is to make the desk-top the focal point, with client/server applications that are proactive rather [than] reactive. This is a major paradigm shift from block-mode terminals to asynchronous message-based computing." Zornes said. "The problem today is that nobody knows how to program in such an environment." To create these environments, application developers will have to familiarize themselves with object-oriented programming tools, because most development environments are not designed to yield these types of client/server applications. . . .

Building a solid base for client/server computing, Avery L. Jenkins & Michail Vizard, DIGITAL REVIEW, 6/8/92

LIBRARIES

As developers move toward object-oriented programming, they are increasingly turning to prepackaged class libraries to reduce programming efforts and to hedge against today's growing number of platform alternatives. However, developers and vendors both agree that their concerns about class library standards and compatibility have yet to be addressed. . . .

Class libraries ease development, Garry Ray, COMPUTERWORLD, 6/29/92

DATABASES

. . . Object-oriented programming and Oodbms are in fact different things. But they share one vital concept—the notion that software or data can be "containerized." Everything goes into a box, and there can be boxes within boxes within boxes. In the simplest programming languages, each step of the program is one instruction; in an object-oriented language, each step might be a whole boxful of instructions. So, too, with object-oriented databases. With one of these, the space that might have been allocated for one data element may in fact be filled with a whole boxful of data. . . .

Here comes Oodbms, Joseph R. Garber, FORBES, 7/6/92

. . . Equally important, as new platforms, such as object-oriented databases, come to market, future-proofed application can be ported to the new platform by changing only the DBMS block—the others remain intact and unaffected. Although the benefits of future-proofing software are numerous and obvious, a major investment of resources is required because there is no way to evolve old mainframe applications to this type of architecture. Noted technology consultant George Schussel, president of Digital Consulting, says, "For maximum advantage from the added power, functionality and richness of downsized client/server approaches, you'll need to trash your old applications and rebuild them from scratch." This is why future-proofed, object-oriented application packages are mostly coming from newer software vendors that have little stake in old application and maintenance revenue.

Commentary: Future-proof technology, Frank H. Dodge, COMPUTERWORLD, 6/29/92

THE BUSINESS OF O-O

. . . This was a particularly funny one, which I didn't hear over the phone but at a conference cockta party. Apparently, the boss—call him Nosbert—was Head of Software in a medium-sized company and the boss's boss—call him Luigi—was Manufacturing Manager in the general corporation and had no real knowledge of software. After reading BUSINESS WEEK on a plane, Luigi called Nosbert into his office. "Nosbert, are WE doing this object-oriented stuff?" "Er, no" "Then get with it. I don't want our company to miss the bus." Nosbert then implemented one of the most painless transitions to object orientation in the history of the universe. He switched his compilers from C to C++ but told his programmers to continue to write C code in the same old

FYI

Excerpts from leading industry publications on aspects of object technology

HYBRIDS

. . . When we examine the choice of developing an expert system with a simple rule-based or a hybrid tool, the hybrid software quickly becomes much simpler to use and understand. Hybrid tools allow us to represent knowledge as objects that have the advantage of mapping directly to things in the real world. As the number of objects grows, we can create hierarchies that show the logical relationships between different types of objects. The objects maintain clarity about the facts being represented in the knowledge base and they make it easy to edit the factual structure of the system. Hybrid tools have an added advantage of allowing those first prototype applications to gracefully grow into more complex applications that would be impossible to develop with a simple rule-based system. At the same time, we reduce the number of rules required, which in turn reduces the memory and processing time reqired . . .

Knowledge-based systems often deal with complex problems that may be poorly defined. They require a network of facts and heuristic rules. Their solution depends on logic and an inference engine that can dynamically create a decision tree, selecting which heuristics are most appropriate for the specific case being considered. The capabilities provided with hybrid tools, combining object-oriented programming, rule-based reasoning and a graphical environment support, rapid prototyping and make it easy to create a high quality, effective user interface. But hybrid tools do more than just combine OOP, ES and GUI technologies. They allow us to solve problems that could not be effectively solved before. In addition, they are becoming very similar to CASE tools in that they provide a descriptive, graphical model of the domain . . . .

Object-based hybrid tools, Ross G. Hopmans, HP PROFESSIONAL, 5/92

LANGUAGES

. . . If Smalltalk is so powerful, why does it have such a small following compared to C++? A number of possible explanations exist. Dan Shafer, author of the book, *Practical Smalltalk*, suggests that Smalltalk is so completely different from any other development environment that the first reaction of procedural programmers is panic . . . Smalltalk's classes and methods are not just a class library but an integral part of the environment that makes up Smalltalk. Everything interacts with everything else. This can be quite disconcerting for the beginner, and the fear of breaking something can often serve as the greatest deterrent to learning Smalltalk . . . Ultimately, we return to the original question: Why Smalltalk? Because you want an environment built around object-oriented programming, not derived from procedural programming. You want an environment that provides extensibility while managing your code. You want the flexibility of an interpretive language in which you can play with and test your code, coupled with the performance of a compiler. You want an interactive debugging environment that lets you inspect and modify your code and variables on the fly with instant results, instead of saving, compiling, and linking between changes.

Why not Smalltalk?, William Scott Herndon, UNIX REVIEW, 5/92

PREDICTIONS

The next generation operating system will have to do much more than just use more memory; it will have to fulfill a shopping list of items for leading edge computing in the 1990s . . . The next wave of computer software will also have to make it much easier to connect computers into networks, allowing electronic mail and sharing of data . . . Finally, the software will have to be "object-oriented," a buzzword that means it will be easier to write programs that are compatible with each other, and easier to update existing programs, making the system quick to adapt to new uses, from communicating with supercomputers to automating your car. Object-orientation is the key to broadening the PC market by finding new places to put computers now that nearly all the desks are filled . . . .

Code Warriors, Robert X. Cringely, WORTH MAGAZINE, 7/92

. . . The object-oriented programming revolution may be the beginning of the biggest programming advance in the history of computers. It may prove to be the software equivalent of the microprocessor, allowing the mass creation of more capable, less expensive software. We say "may" simply because it may also be that object-oriented programming is just the beginning of that revolution and will itself be swept away in a comparatively short time by the new technologies it makes possible. As usual with computer revolutions, object-ori-

at the same time guide their gurus into producing many positive contributions to the organization. The SWAT team is a group rather than an individual guru. This is easier to manage if you are careful in selection of the group. You should remember to recruit members who have interpersonal skills in addition to a technical background because these are the people who will sell object technology to the rest of the staff by winning friends and influencing neighbors.

To be successful, the SWAT team must be given extra resources, higher funding, and twice the time they think they need when they begin their first assignment. If at all possible, involve the team in the evaluation and selection of O-O platforms; if not, make sure they all have documentation, software, and hardware of the highest caliber available, evenly distributed among them. And start training promptly: train from day one, then give them play time and open access to the technology. This first training should consist of bringing a training consultant in-house, at least for several days, to get the team moving. Focus on O-O analysis (OOA) approaches. OOA skills are important and can be used on projects implemented in any language (structured or O-O). It is useful at this point for the independent consultant to offer an overview of current technology vendors and platforms. If desired and if the team feels it would be helpful, the consultant can continue on to a lower-level review of particular languages, tools, or database management system (DBMS) products. I feel OOA skills should come first because they provide the basis for good object-oriented design (OOD) practices when using OOP languages. I use "CPR for OOA" as a start, a variant on CRC cards. These provide a manual approach that can be used easily from the beginning. Once facility in OOA concepts is achieved, more formal methods such as Booch notation are readily adopted. Notice that I do not start with automated tools, although I encourage their adoption later on. It is more important to me that analysts think through the problem. They can always learn to use a specific tool later. During the next few weeks the SWAT team is on its own, engaging in internal education exercises such as reading some of the major textbooks and magazines and following up with group discussions. These discussions tend to be nonthreatening because the team is not yet working on a particular project. During this period, after the initial training, the output should be "samples." That is, the initial team should be free to "play" with the technologies. For example, if you will be using Smalltalk, this is the time for the SWAT team members to begin examining the class libraries. If the decision on ParcPlace vs. Digitalk implementations has not yet been made, team members should look at both versions. Class libraries, both from vendors and third parties, should be evaluated. This will be invaluable as your organization makes the final purchasing decision on platforms and environments. Don't pressure the first team to implement production jobs. You'll be surprised to find that they often go back to a recent project and reimplement it in an enhanced manner, learning much in the process about how different OOP is from their conventional software-development technology. In the next phase the new O-O team can be assigned to a production job. This should be carefully chosen; it should be more than a "toy" system, yet not so complex that it cannot be completed in less than six months. This is important in ensuring positive feedback to team members as well as continuity in the membership of the team. Before production work starts, it is beneficial to formally summarize the results of the study accomplished in the earlier investigations of object technology.

By the time production work starts, the first O-O team will have found its own guru or gurus, who will often start propagandizing the benefits of object-oriented analysis and programming to the rest of your staff. Don't worry—that's part of switching over to object technology. This is the beginning of a slow process of motivation and education that will make the whole shop eager to get on the leading edge, rather than hang back in fear.

Let them know it can be done by encouraging their efforts, giving them the chance to build self-confidence, and, most important, by providing them with the training they need! ≡≡

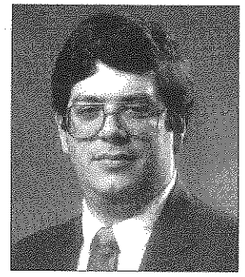
Reference

1. Lorenz, M. Getting started with object technology: effectively planning for change, HOTLINE ON OBJECT-ORIENTED TECHNOLOGY 2(11):9, 1991.

David Bellin is a noted author and consultant engaged in object technology training. His seminar on "Object-Oriented Analysis" is available in-house. Dr. Bellin welcomes readers' comments and may be reached at 919.460.5198 or by email at dbellin@igc.org.

Object-oriented ROI: extending CRC across the lifecycle

(part 2 of a series)



Sam Adams

According to an independent survey conducted by Object Magazine, over 75% of the Fortune 100 have object technology (OT) projects under way. The decision for these organizations is not "Do we go object oriented?" but "How do we maximize the benefits of object technology and manage the risks?" This is the second in a series of articles concerned with the issues of extending CRC to meet the challenges of enterprise-wide computing using object technology.

The previous article defined the requirements for maximizing return on investment (ROI) in object technology by creating and managing reusable software assets. It also introduced the role of CRC in the discovery of key business objects.

This article introduces KSC's lifecycle methodology for the development of object-oriented business systems. Extensions to CRC for entity modeling, requirements acquisition, analysis, and traceability will be discussed in detail. Future articles will deal with extensions for implementation support, multiuser tool requirements for the deployment of object technology on an enterprise scale, and managing software assets using object-oriented metrics.

OT AND ROI

KSC's experiences have shown that maximizing return on investment in OT requires that software be treated as a corporate asset that can appreciate through investment in its quality and reusability. The most valuable software assets of any organization will be the objects that capture the essential nature of their business domain. The foundation of these assets will not only be code but high-quality design information. Pervasive reuse of these software assets must become the norm, but it cannot succeed on a large scale without the existence and proper management of large libraries of software components that are "reuseful" as well as reusable.

IN SEARCH OF A METHODOLOGY

For two years now, KSC has funded a major internal research and development effort aimed at defining a methodology and environment for object-oriented software development, based on our OT work at large business enterprises. No existing method was consistent with our experience in large-scale, successful design and implementation projects. The new methodology had to address the complete software development lifecycle, from requirements acquisition to code validation. It had to provide a single, consistent model for the design, implementation, and reuse of object-oriented software components. Integration was required with host-based legacy systems, databases, and support for client/server distribution. People-centered techniques including CRC needed to be central features. Multiuser environments for the creation, assessment, reuse, and management of high-quality software components had to be developed. A comprehensive suite of metrics for the assessment of design quality as well as code was also required.

PRINCIPLES FOR SUCCESS

To make a system of this scope accessible to the many different people involved in the software lifecycle, the methodology must center on the behavior and interaction of objects and other entities. This perspective provides these benefits:

1. Human beings have many years of experience dealing with complex systems of interacting entities. CRC cards are successful because they allow us to apply our natural skills and experiences to the area of system design.
2. By focusing on behavior and interaction, we can defer implementation-oriented decisions to a more appropriate level in the design process.
3. By limiting the design of the system to the entities and their responsibilities and interactions, the design process is more accessible to the users and system specifiers.

High-quality software is designed to meet or exceed the needs of the user without violating user expectations. Achieving this level of quality requires that software designers apply these principles:

1. Maximize user involvement throughout the process. The user is the best person to determine if needs are being addressed and expectations met.
2. Make the right decisions at the right time. The earlier in the process a design decision is made, the greater is its impact on system quality.
3. Expect and encourage iteration throughout the software lifecycle. Only continual design validation, measurement, and refinement throughout the process can ensure that constant quality management is achieved.

All aspects of the lifecycle, including people-oriented processes, tools, and methodology, must support constant manage-

ADA
Software Maintenance and Development

ADC/AdaScan is now available from Software Maintenance and Development as an option to Aide-De-Camp (ADC), the object-oriented software configuration management system. ADC/AdaScan helps the Ada Developer find which source files belong to a specific program, which major program structures these files contain, and in what order files must be compiled to build an executable program. ADC/AdaScan also enables the developer to identify the type(s) of Ada program units a compilation order represents, such as a specification, body(?), package, procedure, task, or generic.

PO Box 555, Concord, MA 01742, 508.369.7398.

Partnerings & Acquisitions

AI Corp and **Aion Corporation** announced the signing of an agreement to merge. While the two companies will quickly begin the changes required to function as a single operation, the merger will be finalized after SEC review and shareholder approval, which is expected in September of 1992 when a new name for the company will be unveiled. The new company will be headquartered in Palo Alto, but will continue to operate a major facility in Waltham. Robert Goldman, chairman and CEO of AI Corp, will become chairman of the new company, and James Gagnard, CEO and president of Aion, will become CEO.

Digital Equipment Corporation announced an agreement to work with **International Software Systems** (ISSI) on an architectural definition and implementation plan for ECMA PCTE. Digital's marketing and engineering staff in Varese, Italy, which has responsibility for all PCTE programs at Digital, has been contributing to the ECMA PCTE standardization effort for several years. Digital is also a founding member of the PCTE Interface Management Board (PIMB) Association, a nonprofit international organization formed to promote the use of PCTE technology.

Serius Corporation announced the appointment of two key members of its management team. **Wesley Richards** has joined the company as president and CEO, and **James Heffernan** has assumed the role of vice president of finance and CFO.

Software Maintenance and Development Systems entered into a distribution agreement with Seoul-based **Genesis Technologies.** Genesis will market the Aide-De-Camp (ADC) software system and distribute ADC in conjunction with complementary CASE tools and services in the Korean market.

Borland International announced plans to develop and market BRIEF, a professional programmer's editor, and Sorcerer's Apprentice, a network version control system. Borland recently acquired the two professional programming tools from the Solution Systems' division of **Software Developer's Company,** and will now own, develop, and market the products. Under the agreement with SDC, Borland also recruited the core development team of these products for development of future releases.

CenterLine Software announced an independent software vendor (ISV) partnership program called **CenterStage.** Under CenterStage, CenterLine will work with its partners to develop value-added product integrations that complement CodeCenter and ObjectCenter. In addition to receiving documentation and technical support on how to access CenterLine's interactive capabilities, CenterStage members will work jointly with CenterLine to market these solutions.

Phar Lap Software Phar Lap's QuickStart for Windows NT allows developers to use Microsoft's NY tools (32-bit C/C++ compiler, linker, and resource compiler) under MS-DOS. QuickStart is available free of charge from Phar Lap for a limited time. QuickStart allows programmers to start building NT applications without having to wait for their favorite DOS utilities to be ported to NT. To use QuickStart, developers copy the NT tools from the Windows NT SDK CD ROM to their DOS hard disk. QuickStart runs the NY tools by providing a subset of the WIN32 API under DOS, allowing the tools to be run from the DOS command prompt like any other DOS program. **60 Aberdeen Ave., Cambridge, MA 02138, 617.661.1510.**

Tom Sawyer Software Tom Sawyer Software introduced the Graph Layout Toolkit, an automated object-positioning tool providing realtime hierarchical layout services for directed and undirected graphs. When directly bundled in applications, the Graph Layout Toolkit delivers an immediate facelift by dramatically enhancing the readability of graph output. In addition, it allows complete flexibility for multiplatform GUI development. Its many useful design features are accessible through extensible C++ class libraries or an ANSI C

1824B Fourth St., Berkeley, CA 94710, 510.848.0853.

CLASS LIBRARIES

PostModern Computing Technologies PostModern Computing announced NetClasses, a set of C++ class libraries for TCP/IP-based object transport and distributed programming in C++ on Sun workstations. The NetClasses Object Transport libraries allow programmers to move objects between applications using TCP/IP and an asynchronous interprocess messaging paradigm. NetClasses transports generic C++ and National Institutes of Health (NIH)–derived objects as well as NetClasses Typed Objects, runtime configurable objects whose structures are specified by programmers in external files using an abstract syntax notation. The NetClasses Distributed Services libraries form a connection management mechanism organized so that network service providers need not set up explicit port numbers and remote procedure connections (RPC), but rather can simply "advertise" themselves on the network. Agents are active processes on the network that monitor network service advertisements and manage connections between information producers and consumers. The NetClasses Remote Method Invocation libraries allow methods to be invoked programmatically on objects from remote machines. **1032 Elwell Ct., Palo Alto, CA 94303, 415.967.6169.**

ENVIRONMENTS

Interactive Software Engineering ISE's Eiffel 3 is a major revision of the company's object-oriented programming environment designed for large industrial projects, as well as a significant addition of new components. The UNIX-based software will be distributed in components so users invest only in tools and libraries they need for a particular project. The heart of ISE Eiffel 3 is EiffelBench, a programming environment consisting of the Eiffel compiler and a set of tools enabling debugging, browsing, and editing. EiffelBench uses ISE's Melting Ice Technology, incorporating the advantages of both compiled and interpreted environments: efficient code and full static typing, with fast update after changing even large systems, resulting in a fast re-execution cycle. EiffelVision is a high-level graphical user interface library for writing applications for various windowing environments without having to learn their details. EiffelBuild is the interactive application builder with immediate execution. EiffelStore is the class library for interfacing with relational and O-O DBMS, dealing with high-level persistency as well as with storage and retrieval of networks of objects into and from different data bases using the SQL query language. EiffelBASE contains well-designed basic Eiffel libraries, including data structures. ISE Eiffel 3 is available on major UNIX, VMS, and AIX platforms. **270 Storke Road, Ste. 7, Goleta, CA 93117, 805.685.1006.**

C++

Oasys Oasys announced that its Cross 680x0 Tool Kit is available on Silicon Graphics' RISC-based workstations and servers. The Oasys Tool Kit includes the Green Hills C++, C FORTRAN, and Pascal cross-compilers and the Oasys 68K Cross Assembler/Linker. Users will be able to develop applications on IRIS systems targeting Motorola 68040/30/20/10/00, 683xx, and 68881/82 microprocessors. Oasys' Cross C++ compiler is fully compatible with the AT&T C++ specification Versions 2.1, 2.0, and 1.2. It has been validated using the Perennial C++ test suites to ensure compatibility with commercial C++ class libraries and the AT&T specification. The Cross C++ compiler includes the Cross C compiler, allowing developers to transition from C to C++ more easily. All the languages are interlanguage callable allowing users to call C from C++, FORTRAN from C, and so on. **One Cranberry Hill, Lexington, MA 02173, 617.862.2002.**

ment of the quality and value of deliverables produced. These include all forms of analysis and design information as well as program code. Testing and user validation of designs at all stages of development, not just at the level of program code, is also necessary. In an industry where reuse must become the rule instead of the exception, a software methodology that does not explicitly support quality assessment and management is simply insufficient for the task.

CRC, ENTITIES, AND REQUIREMENTS

The CRC technique is highly effective for high-level object-oriented analysis and design. But software development in large enterprises requires much more than design. System requirements must be acquired and analyzed. Dependencies must be defined between the application software being developed and external entities such as users and databases. Frameworks and other design architectures must be discovered, refined, and reused. At the implementation level, message interfaces must be specified. Code must be developed and tested. User interfaces must be designed, implemented, and user tested. And in large, computing-dependent organizations, all these processes must be managed in a multiuser, version-controlled environment. In this article we will focus on the front end of our lifecycle methodology and discuss requirements acquisition, analysis, and traceability.

Before any software is developed, we should always have a clear understanding of what behavior is expected of the application. Easier said than done! Designs are developed to meet requirements and applications are implemented according to those designs. No system can ever be expected to satisfy user requirements if the user's needs are not captured sufficiently and communicated completely. Unfortunately, the very nature of human communication is fuzzy incompleteness, with a heavy dependence on assumed background knowledge. Our experiences using CRC with groups of users have shown that requirements are never perfect. They evolve throughout the analysis and design process until they provide a sufficient basis for a successful design and implementation.

One of the many artifacts produced by "waterfall" methods is the notion that all requirements must be specified completely before any design or implementation begins. But it is far more important for the system to meet user expectations upon delivery. Requirements and software actually evolve together throughout the lifecycle of the project (Fig. 1). In the iterative approach to software development, users stay "in the loop," refining their requirements as they better understand that application features are possible within the budget allowed. As an added bonus, users become fully vested in the development of "their" application.

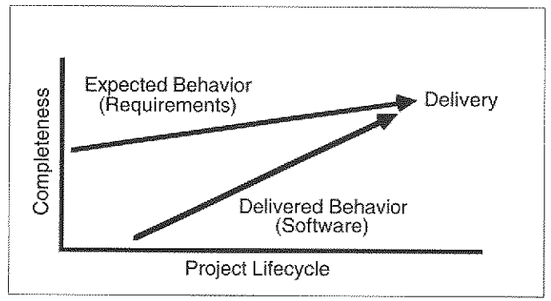


Figure 1. The coevolution of software and requirements.

REQUIREMENTS ACQUISITION

The key to acquiring high-quality system requirements is to maximize involvement of eventual system users and other domain experts in the process and capture their needs in a form that will support the many different activities in the development process. Rather than being clinically observed and analyzed like creatures swimming in a drop of pond water, they should be full participants in defining and validating their system requirements.

Users are often not concerned with or even aware of the application software objects that are created or reused during the analysis and design process. They are concerned with the responsibilities to be performed for the organization as part of their job, and they want or need a computer application to assist them in fulfilling those responsibilities. As software designers, we know that these system requirements must eventually be fulfilled by some group of software objects in the implementation. At this phase of the process, however, it is much more important to understand the user's view of what behavior the system must provide.

FROM OBJECTS TO ENTITIES

Modeling the world in terms of "objects" is a natural perspective taken by users in a CRC session. The word "object" is quoted here because early in CRC sessions it is often unclear which cards describe software objects and which describe other behavioral entities that interact with the software. This ambiguity allows us to focus on understanding the essential behavior of the domain under investigation. The decision as to how to deliver this behavior can be deferred to later in the design process. Often some "object" is created to model the user, at least as far as interactions with the proposed system are concerned. It is also common for an "object" to be created that models the behavior of some part of the system larger than a single object or class. Frameworks such as windowing systems or components that provide relational database interfaces are examples. Based on these observations, we have generalized the CRC concept of a class with responsibilities and collaborators toward the broader notion of interacting behavioral entities.

We did not choose the term "entity" lightly. Fully aware of its usage within the database and information modeling communities, we specifically chose the term to help the transition of thousands of business programmers already familiar with information modeling concepts towards object technology. Designing systems using abstraction and inheritance is not unique to object-oriented approaches. Database designers have been developing both data models and information models using these techniques for decades.

For example, entity-relationship (ER) models are rich sources of information about the enterprise, its business rules, and the na-

ture of the business as currently practiced. Because of the data abstraction and type inheritance common in these models, it is easy to confuse them with object-oriented designs. But although these primarily data-oriented designers did not produce "real objects," many if not most of the "data entities" existing in today's information models provide reasonable starting points for the transition toward fully object-oriented enterprise models. Adding behavior to data entities is not automatic or easy, and significant refactoring of the design is often required to fully take advantage of the additional benefits of the object paradigm.

Evolving data and information models to object models is a broad topic and deserves fuller treatment in a future article. For now, consider an entity as any behavioral unit that can interact with other entities to achieve some goal. Examples of entities that might appear in a design are users, user interfaces, frameworks, database managers, host or network-based application servers, and, of course, business objects such as Employee, InternationalCurrency, and LifeInsurancePolicy.

MODELING REQUIREMENTS WITH RESPONSIBILITIES

It is natural in CRC sessions for users to view "the system" as a single behavioral entity. Therefore, it should also be natural for users to describe requirements for the system similarly to describing responsibilities of an object in a CRC session. But while the essential, high-level behavior of a system may be initially described in terms of concise verb phrases, a much richer model for responsibilities is needed to serve adequately as requirement specifications.

Over the years, many different models for requirements have been used in the computer industry, each with its own advantages. In our experience, one of the most effective ways to describe requirements is in terms of needs and scenarios. According to this model, each requirement has a statement of the need to be fulfilled. It also has one or more scenarios (use cases) that both enhance the understanding of the needs statement and provide a set of acceptance criteria to help determine if a proposed design or implementation alternative actually fulfills the requirement.

Since responsibilities in CRC already have something similar to a needs statement, we can enhance them by including scenarios to support the modeling of requirements. As an added benefit, the creation and testing of scenarios that previously occurred informally in CRC sessions can now be formalized and supported directly in CRC. With these extensions, CRC can now be used to find the requirements (responsibilities) of either objects or entities, using scenarios to test alternative designs.

REQUIREMENTS ANALYSIS

The scale and complexity of most business applications requires that we analyze and design the behavior and interaction of larger units in the model that are not intended to end up as individual object classes in software. By using single CRC cards to represent entities like entire host systems and geographically distributed business units, we find that the same successful results can be obtained as when CRC is used to define software objects. This has allowed our customers to develop more complete and detailed models of not only their users' needs but also the needs of their organizational units. An added benefit is that these models provide evidence for redefinition of the user's responsibilities, which has led to some surprising and profitable insights for several of our clients.

REQUIREMENTS TRACEABILITY

Traceability is about determining whether or not all requirements have been met and completely implemented, as well as deciding which parts of design and implementation are involved in fulfilling each requirement. Aside from determining when a system is "finished," this information is very useful for determining the impact of proposed changes to the requirements, design, or implementation.

By using behavior and interaction to define both software objects and the application components they support, we can trace requirements throughout the entire development process. Like software objects, entities now encapsulate their implementation details by providing an external behavioral interface composed of the responsibilities they agree to fulfill for their clients (Fig. 2). The terms *client* and *server* are used here to indicate service requesters and service providers rather than workstations and host mainframes in the distributed computing sense.

Each responsibility is fulfilled using some collection of collaborating entities acting as servers, along with a mechanism (procedure) that defines how these entities interact with each other to deliver the specified client behavior. These mechanisms provide a high-level "implementation" for each client responsibility, linking delivery of that behavior with the proper implementation of responsibilities provided by the collaborating server entities. This link between high-level behavior and lower-level

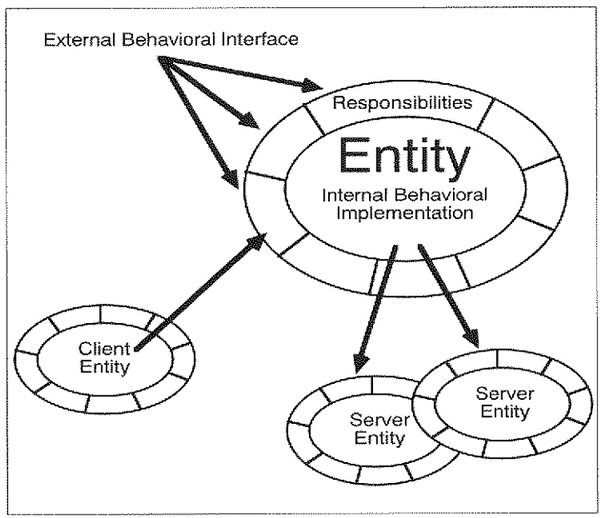


Figure 2. Behavioral encapsulation in entities.

Product Announcements

Product Announcements is a service to our readers. It is neither a recommendation nor an endorsement of any product discussed.

DATABASES

Expertek

Expertek's Zoom is a minimalist single-user OODB that delivers a large fraction of full OODB performance (including cached, keyed random, or sequential access) at a small fraction of the price. Expertek supplies source code compatible with Smalltalk/V DOS, 286, Mac, PM, and Windows. Readable source in Smalltalk/V allows you to enhance or translate Zoom to meet your needs. In addition, telephone, modem, and mail support is provided. **P.O. Box 611, Clatskanie, OR 97016, 503.325.4586.**

Enfin Software Corp

Enfin announces that both 16-bit ENFIN/2 and 32-bit ENFIN/3 object-oriented development environments have added support for IBM's Distributed Database Connection Services (DDCS/2). The IBM DDCS/2 gateway allows DB2 and AS/400 databases to be accessed from a PC as if they were OS/2 databases. DDCS/2 implements IBM's Distributed Relational Database Architecture (DRDA) for access to data in supporting database management systems. Both ENFIN/2 and ENFIN/3 allow an application developer to interactively design and create GUI screens and reports and link the GUI objects to external databases. Consequently, GUI front ends for DB2 and AS/400 can be quickly created with a minimal amount of programming. However, since the ENFIN tools automatically generate Enfin Smalltalk source code, applications can be further customized and extended as necessary. By supporting the 32-bit architecture of IBM OS/2 2.0 operating system, ENFIN/3 performs substantially faster as a development tool, as do the resulting applications. Enfin Software will also continue to provide new releases of ENFIN/2, its 16-bit version for OS/2 Version 1.3, throughout 1992 so that existing users will be able to migrate to the OS/2 2.0 platform as they wish. The release of ENFIN/2 for Windows 3.1 supports virtually all of the new Windows 3.1 features, including OLE. **6920 Miramar Road, San Diego, CA 92121, 619.549.6606.**

Answer Software

Answer Software's HyBase V3.0 is a database server for the Macintosh, with a data model that combines the best features of relational and object-oriented technologies. The relations capabilities allow you to create tables and access your data using SQL operators like SELECT. The object-oriented features let you define your own data types and program methods for operating on them. HyBase allows you to combine these two models to create powerful database applications. The HyBase server runs under System 7 or under Multifinder in System 6. Client applications may communicate with a server on the same machine, or on other processors over an AppleTalk network connection. Answer Software provides a client API for developers who are writing standalone applications and a client XFCN interface for developers who prefer to use front-end tools like HyperCard, SuperCard, or PLUS. HyBase supports a Pascal-like programming language augmented with object-oriented and SQL operators. HyBase statements can be executed interactively or compiled for faster processing.

20045 Stevens Creek Blvd., Cupertino, CA 95014, 408.253.7515.

TOOLS

Iconix Software Engineering

Iconix Software is shipping ObjectModeler, a CASE tool that supports object-oriented analysis, design, and object-oriented programming in a single module. ObjectModeler is available onthe Macintosh and will soon be available on other platforms. It supports the object-oriented analysis method developed by Peter Coad and Ed Yourdon as described in their 1991 book, Object Oriented Analysis. For design, it supports the class and object diagrams in the extensions of the method developed by Grady Booch and described in his 1991 book, Object Oriented Design With Applications. ObjectModeler also supports object-oriented programming in C++, with a C++ language-sensitive editor. The tool is multiuser enabling concurrent repository access with collision detection, access controls, and global functions across a network. ObjectModeler is fully integrated with the nine other modules in the Iconix PowerTools set.

2800 28th St., Ste. 320, Santa Monica, CA 90405, 310.458.0092.

OBJECT-ORIENTED SOFTWARE ENGINEERING
A Use Case Driven Approach

reviewed by
Brian Henderson-Sellers

I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard
Addison-Wesley/ACM Press, 1992
524 pages; ISBN 0-201-54435-0

The primary author of this book, Ivar Jacobson, is well known and highly regarded in object-oriented circles. His approach and his methodology, ObjectOry, have been sparsely reported in the literature until the publication of this long-awaited volume.

The ObjectOry methodology for object-oriented analysis and design has had many years of industrial exposure: 15 full-size projects have used it successfully. While many of its characteristics are now evident in other approaches, the most significant novelty is the emphasis on what Jacobson calls *use cases*. A use case is a sequence of transactions performed by a user of the system. Each use case outlines a likely thread of control through the many objects within the system, thus providing a potential unification between the static and dynamic views of an object-oriented system—an integration lacking in most other methodologies. It is further argued that by structuring methodologies to build systems around action patterns of likely users, a greater degree of reusability and flexibility will be built into the system

The book is divided into three parts: The first is introductory; the second is the meat of Jacobson's methodological argument; and the third ranges over two detailed case studies, some project management issues, and a comparison with other methodologies and notations. In his preface, Jacobson outlines alternative reading strategies for various types of software engineers and managers, allowing the text flexibility to meet the needs of different audiences.

The introductory part of the book is well presented in five chapters beginning with an easily understandable analogy to the building trade, with its various levels of abstraction, quality concerns, reuse, etc. The second chapter considers systems lifecycle issues while the third is a synopsis of what is generally understood by the object-model. The book takes the reader from zero knowledge about object technology to fully detailed methodologies (notably ObjectOry). In that sense, the book's real audience is not novices but initiates. Chapter 4 applies the ideas of Chapter 2 to object-oriented specifics and Chapter 5 addresses languages.

The focus of the book in Chapters 6–12 is ObjectOry. Jacobson presents a trimmed-down version of the method, which he calls OOSE (object-oriented software engineering), because he claims the full description of ObjectOry is too massive to be contained within a text of this kind. Nevertheless, the reader will come to understand the basic philosophy of ObjectOry here. It is also stressed in the two forewords to the book by Dave Thomas and Larry Constantine, who both strongly endorse Jacobson's work.

These chapters take us through the lifecycle that Jacobson sees as a waterfall for each version. I heartily concur with his discussion of analysis modelling, with its problem of space focus and modelling underpinning. The translation to design is done notationally as well as via the acquisition of a new mindset. Analysis icons are transmuted into design block icons that are abstractions of the implementation and may be coded as one or several classes. In this text, design and implementation are coupled under the heading of "construction" (Chap. 8). The following four chapters in Part II consider, in relatively brief detail (about 15 pages per chapter), the subjects of real time, database, component generalization and reuse, and testing strategies.

Part III is named "Applications." Of the four chapters here, the first two certainly fit under this heading. However, the third deals with managing object techology. Since one of the intended groups of readers is senior management, it's a little puzzling to see this important chapter "tucked away" almost at the end of the book.

The final chapter contrasts ObjectOry/OOSE with other published methodologies, although the discussion tends to be biased toward a description of the models of each methodology rather than the process of building the models. This illuminates the novel concepts embodied in Jacobson's work.

The book has two appendices: one on the historical development of these ideas embodied in ObjectOry (where you may find what CASE tool support is currently available for this methodology) and a summary of the architecture.

The book is a very solid software engineering book. It is not an explanation or advocation of the advantages of object technology but it does assume that objects are useful and describes a whole software engineering environment focused on this new technology. This perspective suggests a longer life than other texts on object-oriented concepts that perhaps more directly reflects today's manifestations of object technology. Jacobson and colleagues have, I believe, captured the essence of object technology in much greater depth: that of software engineering and not just software development.

I did notice some disturbing repetitions in the text. For example, Figure 7.12 is identical to Figure 6.14 and some of the tabular material in Chapter 16 also tends to be repetitious. The book's production is generally excellent with only a small number of typographical errors, although there are too many instances of tables and figures located several pages away from the textual reference, sometimes even in a totally different section of the book. However, the references are reasonably complete and the index is certainly adequate.

This book is highly recommended, in part for managers and as a whole for technologists coming to grips with object technology. While the terminology used by Jacobson may appear strange to some at first, largely because this material has been developed parallel to and often earlier than the terminological standards emerging elsewhere, the authors' understanding of object technology shines through. ≡≡

Brian Henderson-Sellers is with the School of Information Systems, University of New South Wales, Sydney, Australia. He can be reached at brianhs@usage.csd.unsw.ozau.

behavior provides traceability in both directions. As an example, let's revisit the mail order company CRC card described in last month's article:

MailOrderCompany	
publish catalogs	CatalogPublisher
convey product offerings	ProductList
maintain product inventory	Warehouse
convey address and phone number	
sell products	Customer, Salesperson

Of these responsibilities, "publish catalogs" and "sell products" are the principal external behaviors. Figure 3 shows the principal interactions between the MailOrderCompany, its Customer(s), and the PostOffice.

In taking a closer look at the responsibility "sell products," we find that this high-level behavior is fulfilled by a number of other entities collaborating together (Fig. 4). Some of these entities will eventually be implemented as software objects, while others, like Salesperson, may be used to (re)define the job responsibilities and work procedures for a human salesperson. In the resulting "system," Salesperson's behavior may be delivered by a person performing a job function manually, or by using a software application with software entities collaborating with him. Which entities are automated and how they will deliver their behavior are decisions that can be deferred to later in the design process.

While we are only showing one mechanism for "sell products," it is common in object-oriented designs for several alternative implementations of the same responsibility to be available. In this example, the Customer is making a credit card purchase. There might also be other mechanisms for "sell products" in MailOrderCompany that address different forms of payment or even different kinds of sales, such as quantity discounts and government purchase agreements.

If we take an even closer look, this time at the responsibility "check credit limit" in the Purchase entity 1145, we find also find a mechanism (Fig. 5).

If we decide to implement this entity as an object in Smalltalk, for instance, a specific method in the class Purchase uses program code to fulfill the responsibility. The Smalltalk code shown in Figure 6 is actually an executable version of the mechanism shown in Figure 5. It should be noted that interactions 1 and 2 in Figure 5 represent the Purchase entity 1145 using its own responsibilities "determine" total price and "determine total tax."

Tracing requirements and their fulfillment for interacting responsible entities is possible because of the association of responsibilities with the mechanisms or source code that describe their implementation. In Figure 3, the "sell products" responsibility is involved in completing the interactions between the Customer and the MailOrderCompany. The dashed rectangle indicates the part of the diagram shown in greater detail in Figure 4, which illustrates a mechanism for the implementation of "sell products." The dashed rectangle is also shown in more detail in Figure 5, which details a

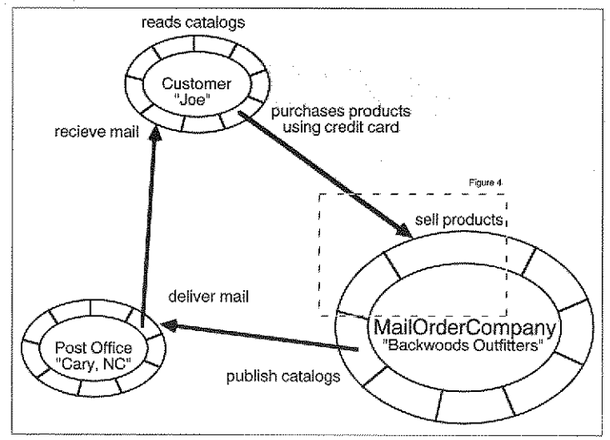


Figure 3. High-level interactions between entities.

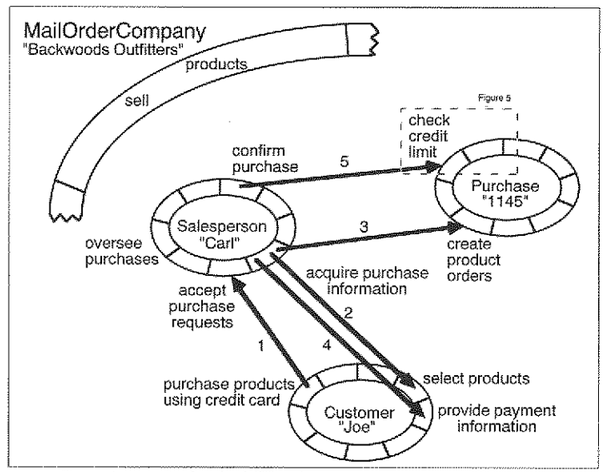


Figure 4. A mechanism for fulfilling the responsibility of a high level entity.

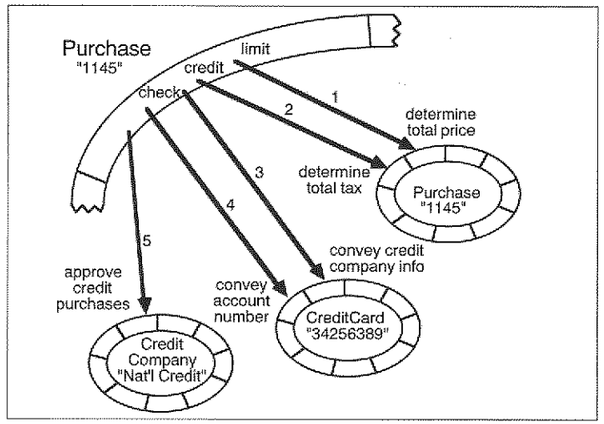


Figure 5. The mechanism for an object's responsibility.

mechanism for the implementation of the 'check credit limit' responsibility in the Purchase entity.

These diagrams illustrate the connection between the implementation of one entity's responsibility with those of the entities and objects involved in the mechanism. This connection completely integrates the behaviors of lower-level and higher-level entities and eventually the executable software itself. A change in the behavioral interface of the Salesperson entity in Figure 4 might invalidate its role in the mechanism for "sell products" in MailOrderCompany. By replaying the scenarios that test the mechanism, the now-missing behavior would be detected, resulting in either its reinstatement in Salesperson or the development of a different mechanism for "sell products."

THE REST OF THE STORY

The large amount of interwoven information required for business system designs, whether object-oriented or not, must be managed throughout the lifecycle to protect the organization's investment in its reusability and value as a software asset. In addition, version control and configuration management of all relevant information is mandatory if large group design and development is to be managed with confidence. For this reason, multiuser design and development environments are required to fully support the high level of integration provided by our methodology.

checkCreditLimit

"Answer true or false depending on whether or not my customer has sufficient credit available to buy the products in the quantities specified."

```
| totalCost creditCompany approval|
totalCost := self totalProductPrice + self totalTaxForPurchase.
creditCompany := CreditCompany
                named:self customer creditCard companyName.
approval := creditCompany
                approvePurchaseAmount: totalCost
                forCustomerAccount:self customer accountNumber.
^approval
```

Figure 6. Smalltalk method implementation.

The next article in this series will present an overview of the design and implementation portions of our methodology, along with a detailed discussion of the software environments required to support the enterprise software lifecycle from requirements to code and back again. ≡ ≡

Sam Adams is the Senior Consultant and founder of Knowledge Systems Corporation. Since 1984, Mr. Adams has been actively developing object-oriented software systems in Smalltalk and is widely recognized for his expertise. He is codeveloper of a group facilitation technique using CRC cards and has been training professionals in object-oriented technology for over six years. Mr. Adams has served on several conference committees and is a frequent speaker and panelist at leading industry conferences. He can be reached by phone at 919.481.4000 or by fax at 919.460.9044.

STRAGETIC BUSINESS SYSTEMS ≡ ≡

Continued from page 1

in terabytes, the insurance company IS manager won't be excused because the object-oriented database "can't store an object that spans physical volumes."

The world of the IS manager is more demanding than many realize. It requires saying in advance how long it will take to develop a new system, and being within shooting distance of that estimate when the results are in. It often constrains solutions by requiring the use of equipment and people it already has (no—we can't "shoot all the COBOL programmers" as one industry executive once suggested). And it *must* find new technologies to help it out of the mess it's in.

The question is whether object technology is *the* technology and, if so, how we can introduce it to the million or so programmers and analysts who call the IS organization their home. Of course, one could argue that object technology was never meant for creating large-scale business applications, that it's most applicable to the creation of complex user interfaces and workstation-based applications. But I think our industry would be missing the boat. The concepts of object-oriented design are certainly as applicable to difficult business problems as they are to difficult technical problems and complex user interfaces. At AMS, as well as at a number of other companies, we are actively engaged in bringing object technology up to industrial strength for use in stategic business systems. There are some difficult problems being addressed in a way that makes object technology more useful in this arena.

Of course there are always counterexamples: "high-volume, industrial-strength systems" have already been developed using object-oriented techniques, but these are typically embedded systems, operating systems, or development tools.

STRATEGIC BUSINESS SYSTEMS AND OBJECT TECHNOLOGY

I define a strategic business system as one that is integral to the day-to-day operation of a large organization. Today, such systems are typically characterized as monolithic, COBOL based, and mainframe oriented. More and more, however, they are being "downsized" to networks and distributed architectures. Data integrity is paramount and hours-long batch-processing runs are common. The systems are changing from text to graphics but they are still the same behemoths necessary to run today's large organizations.

A strategic business system's portfolio may encompass hundreds of systems with total code in the tens of millions of lines. The limiting factor in improving IS, therefore, is often the management processes we have or are capable of developing. Design management, configuration management, and versioning are qualitatively different with a project team of 100 people. When dealing with such projects, the management processes cannot be left to chance or informal methods. They need to be standardized, documented, and supported by appropriate automated tools. These barely exist today for projects employing structured and information engineering methods, let alone projects using object technology.

So far the vendors of object technology have given us, the users, only pieces of a complex puzzle. We are unable to see what the completed picture will look like—or even if the pieces will fit together without holes! Each vendor has its niche, each book its topic, each speaker an ax to grind. But will it all work when we try to develop strategic business systems for today's large organizations?

We can't really know. With only one or two visible examples (implemented several years ago using specially developed tools,

volvement of people outside the project development team. This involvement is crucial for keeping the work of your O-O team current with mainstream company objectives.

UPDATE THE FIRM'S REWARD AND RECOGNITION SYSTEM

This is a definite "sticky wicket" for OT. Think carefully about the goals most important for your project and your company. Create rewards that reinforce these ideals. Reward systems based on performance in previous-generation software development are almost always wrong for rewarding the best performance in OT.

Here's a simple approach used in one project: programmers new to OT are given some training and then assigned to write methods for classes that have been designed but not yet implemented. The project leader does the bulk of the system design including the inheritance hierarchy and class invariants. Only after programmers prove they can perform adequately at this assignment are they allowed to advance by taking on inheritance decisions. They gradually take on more and more responsibility for design and coding decisions until they eventually become responsible for an entire subsystem. This is a great common-sense approach to reward and recognition that can reinforce your O-O training program.

VIEW MISTAKES AS LEARNING EXPERIENCES

Mistakes represent opportunity. If we made no mistakes, there would be no way to improve our processes and we could not look forward to competing at higher levels of efficiency. This reality should be acknowledged in O-O design by applying the principles of programming by subcontracting. Assumptions should be codified via class invariants, preconditions, post-conditions, and other types of assertions. This is directly supported by Eiffel and can be implemented in other languages as well.

> ***"The O-O enterprise model is really the software embodiment of the concept of ever-increasing improvement demanded by TQM."***

The cynic might complain that all the extra assertions are just additional opportunities for mistakes. While this is true, they typically prevent far more mistakes than are added. Our own experiences with Eiffel have shown that runtime defects can be decreased by almost an order of magnitude. (Other aspects of Eiffel such as automatic memory management also helped out in this regard.) I have also used assertion technology with C++ and estimate that runtime problems were reduced by perhaps 25%.

EMPOWER EMPLOYEES

Don't neglect the opportunity to involve as many people as possible in the O-O process. O-O is really a way of thinking, so expose the concepts to your best thinkers and let them help you. Doing so can ensure that your ultimate success with O-O is their success as well. Realize that O-O gives you organizing principles that allow better understanding of the work your department performs. In this way, O-O becomes a bridge to the rest of the company.

Once enough key people in the company are aware of basic O-O principles, the opportunity exists for building an O-O enterprise model. This model can become the tool whereby management and employees simulate and ultimately gain control over large-scale corporate processes. Of course, the very act of attempting to create this model will uncover areas of needed improvement. The O-O enterprise model is really the software embodiment of the concept of ever-increasing improvement demanded by TQM.

CONCLUSION

Making the switch to O-O is a difficult task. Treat it with respect. Remember that the only thing we can be sure about in this industry is constant change. If we can deal with that, everything else is easy. While better approaches to software development like OT are an important part of the changing picture, we also need to install process-review mechanisms and other quality-focused practices to help us on the road to ever-improving software development. ≡ ≡

Suggested reading

1. Gitlow, H.S. and S.J. Gitlow. THE DEMING GUIDE TO QUALITY AND COMPETITIVE POSITION, Prentice Hall, Englewood Cliffs, NJ, 1987.
2. Peters, T. THRIVING ON CHAOS, Alfred A. Knopf, Inc., NY City, 1987.
3. Eisman, R. Why companies are turning to total quality, LEADING EDGE EXECUTIVE, May 1992.
4. Meyer, B. OBJECT ORIENTED SOFTWARE CONSTRUCTION, Prentice Hall, Englewood Cliffs, NJ, 1988.
5. Howard R. Eiffel at the Georgia Tech College of Computing: An interview with Dr. Brian Guenter, THE EIFFEL OUTLOOK JOURNAL, July/August 1992.

Robert Howard is President of Rock Solid Software Inc. in Austin, Texas, which sells and supports tools and libraries for the Eiffel programming language. He is editor and publisher of Eiffel Outlook, the leading independent technical journal for the international Eiffel community. He can be reached by phone at 512.328.6406 or by email at rock@rock-sld.com.

software engineers. Evaluation of these processes must be ongoing and address all of the following areas:

- standards for design capture documents
- coding standards
- guidelines for creating rough and fine-grained work estimates
- guidelines for evaluating outside libraries and tools for possible acquisition and use
- guidelines for test plans and the testing process
- guidelines describing the typical product release mechanisms
- a process definition of how improvements in reusable libraries are maintained and redistributed

ESTABLISH EMPLOYEE TEAMS

On each large project, an employee team should perform the following tasks:

- Create the highest-level cut at the interaction of the key proposed objects in the project (OOA).
- Suggest the best opportunities for reuse of existing software.
- Review the first proposed inheritance structure proposed by the project engineers.

> *Our experience suggests that project scheduling estimates can improve drastically with the adoption of OT.*

Since O-O analysis is done in terms of the application domain, it is necessary to have domain experts as team members. In fact, domain experts should be fully involved in design decisions throughout the project. (A useful hint: team meetings that concentrate on O-O design are an excellent opportunity for demystifying OT for all types of observers. Use them to expand your base of O-O cognoscenti.) After each major phase of a project is completed, it is important to perform a postmortem analysis of the project:

- Locate where the basic processes either broke down or were improved by the project members.
- Review the project estimates. Which were right? Which were wrong? Why? How could the estimates have been improved?
- Was anything else of value learned? Both positive and negative lessons should be included.
- Identify and note candidate code for possible reuse at a later date. (I don't suggest taking the time to make this code fully reusable until a specific opportunity for reuse has been identified.)

Findings are only valuable when shared with the rest of the organization, so be sure they are reported to the quality team.

SET UP SPECIFIC MEASURABLE SHORT- AND LONG-TERM GOALS

The key word is measurable. While we don't yet have standard O-O metrics that capture the dynamics of the O-O lifecycle, we can still measure many things that loosely represent our progress. The two most important bottom-line measurements are defects per release and a measure of the accuracy of project time and effort estimates. The measurement of defects is conceptually simple because bug tracking is a normal part of almost all software projects. The difficulty comes when one ruthlessly notes all types of defects including design shortcomings, holes in the requirements analysis, and even inclusion of unnecessary or useless features.

The accurate measure of project estimates requires that the initial estimates (which always change) be retained for later study. Let the developers know that this will happen. Point out that time and cost estimation are meaningful components of any engineering discipline.

Our experience suggests that project scheduling estimates can improve drastically with the adoption of OT. Here are some hints: Allow a first cut at the object design before asking for estimates. The best estimates I've seen were those derived by summing subestimates of major methods identified within the initial object design. Also obtain confidence levels for the stability of the initially proposed inheritance trees. Current O-O languages and tools typically do not allow the inheritance hierarchy to be changed easily. Inheritance modifications are usually conceptually easy to perform but are often time consuming; factor in extra time if confidence levels are less than 95%. Finally, increase all estimates for new classes and methods that will be reused by more than one programmer during the course of the current project. Design reviews, additional testing, knowledge sharing, and intraproject maintenance will eat up more time for these than for code that is not reused in this manner.

SOLICIT FEEDBACK FROM CUSTOMERS

A rapid-prototyping approach to development, where a working executable version of the program exists throughout all stages of development, is a big win because of early feedback. The problem is finding someone to provide the feedback. The ideal scenario, which some say is practically a requirement, involves a product champion who performs this service. You can also use the working prototype as another tool for broadening the in-

techniques, and languages), several issues fundamental to the IS world have not been adequately addressed, namely, scalability, manageability, and performance.

The entire field of software engineering generally has evolved in response to these issues. Software engineering provides the systematic approaches and standards that enable us to develop large systems, manage large projects, and achieve acceptable performance. Without this discipline, we produce spaghetti code that costs a lot to maintain (or becomes unmaintainable) and is unable to meet the changing needs of the organizations it is meant to support. I hope we have not forgotten this in the rush to develop new systems by "iterative development," create reusable software based on object libraries, and get to market first with new methods, tools, and languages.

SOFTWARE ENGINEERING FOR OBJECT TECHNOLOGY

For the past 25 years, our industry has labored to evolve into a profession worthy of the name "software engineering." Over that time, many aspects of a discipline for software engineering have been explored. The key aspects that need to be addressed for object technology are:

1. a defined lifecycle (phases and tasks)
2. methods and techniques for performing those tasks
3. tools and languages
4. a "design repository"
5. estimating methods for development effort and cost
6. estimating methods for operating costs of systems
7. training and educational curricula

Norman Plant's article in the HOTLINE (3[7]) provides specific requirements for many of these areas, but I would like to address a few in terms of three fundamental issues.

Lifecycle

A lifecycle for developing large object-oriented information systems must allow for the orderly formulation of system requirements, translation of those requirements into tested, high-performance software, and integration of components into libraries that can be used in other systems.

Rebecca Wirfs-Brock's article in THE SMALLTALK REPORT[1] lays out some ideas for such an object-oriented system lifecycle. With a few added components, her proposal sounds suspiciously like the development lifecycles being followed by most organizations today. Perhaps this is because a lifecycle designed to produce an industrial-strength system is by nature more structured than one designed to develop a prototype or even a small-scale system. Her lifecycle begins with the process of developing an overall concept and design for the system before getting into the iterative, object-oriented programming cycle. It ends with explicit "cleanup" and "generalization for broader utility" (as lifecycle phases) as necessary postimplementation steps. In short, she attempts to fit the iterative prototyping structure of the object-oriented development environment to the lifecycle of industrial-strength systems based on reusable components.

Methods and techniques

Within the context of the lifecycle, methods and techniques must be specified for how to go about the design, development, and implementation of object-oriented systems.

You have only to visit your local technical bookstore to see the number of object-oriented design methods that have been proposed. While there are similarities, each has its own approach. Very few have been validated, particularly as they pertain to large systems. This is currently a major source of confusion for organizations attempting to move into the object arena.

Methods and techniques usually imply documentation. And documentation, to be readily understandable, needs to have a standardized form. Experience to date seems to show that object-oriented systems have an even larger requirement for documentation and comments in the code than 3GL-based systems. Most of us know the importance of reading in English what the original programmers thought they were doing and why. The object technology goal of reuse actually brings with it a higher standard for documentation quality.

Tools and languages

Software engineering tools and programming languages provide the medium for applying methods and techniques to the production of operational systems. I am not referring to just programming tools, but design tools, testing tools, configuration management tools, etc.

For 25 years, IS systems have been coded primarily in COBOL. There was, for a while, a flurry of activity in PL/1 but it never really caught on. (I wonder what we can learn from the fact that the one highly publicized object-based business system in existence today was implemented in PL/1.) Now we are being asked to switch to C++ or Smalltalk, languages whose richness is constrained not so much by their syntax as by the class libraries that are really part of the "language environment." Studying the base classes is really "learning the language" and we're not much better off than we were 10 or 15 years ago with COBOL: programmers and designers still have to study the specific dialect and class libraries of the particular implementation they want to use. Having a consistent set of base classes across the different implementations of Smalltalk and C++ will make it a lot easier to develop a cadre of experienced object developers. If the differences remain significant for too much longer, they will impede the transition to object technology by making it too difficult to move people from one environment or project to another.

The biggest tool challenge for object technology, however, may be creating the kind of "object library browser" that will integrate indexing, documentation, and code in a way that truly makes large-class hierarchies accessible. True, there is a significant compactness of code in object-oriented programming languages, and significant reuse will also account for reductions in lines of code, but by how much? Even at a 10-to-1 reduction from today's systems, providing equivalent functionality will mean systems with thousands of objects and tens of thousands of methods. Since a large part of using object technology is understanding the class hierarchy of the system and thus understanding what can be reused, how will we approach design and development of such massive

systems? At what point will "dis-economies" begin to set in? A 100-object system can be understood in detail by a single person, but what about a 1,000-object system or a 10,000-object system? This, of course, assumes that we will figure out how to document and index objects in a meaningful way when we have thousands of objects.

Other tools also need to be scalable. One of the configuration-management tools we have been using to help control a Smalltalk development effort gets high marks for capabilities but seems to have been designed with a different size project in mind. Each "application" defined to the tool is limited to about 50 object classes. When more than five people are working together, the disk files get so large that you have to reorganize every other day. The ideas are good, but this type of tool will need a major overhaul in features as well as performance to support a project ten times this size.

The key point here is that large business systems will exist for 20 years or more. We need tools and languages that will be effective during development and during the 90–95% of the system lifecycle that occurs after the system goes into production.

Design repository

A design repository provides a generalized model for the information concerning a system that must be stored as a basis for future enhancement and maintenance. It provides a common facility for tool developers to store the information their tools create and for tools users (i.e., application developers) to access this information.

Object-based systems will need this type of repository as much if not more than traditionally developed systems. The use of object technology does not relieve us of the need to access a wide array of information as we perform maintenance and enhancement. In fact, the fundamental ideas of reusability demand this access. The object hierarchy itself contains some of the information that one would store in a design repository, but it is neither sufficient nor in a form that can be easily accessed.

We all know the problems that IBM and other CASE tool vendors have had in trying to agree on a common repository of information that can be created about application systems. In this as in other areas, the advance of object technology will be hindered by the unsolved general problems facing our industry.

Estimating methods

Estimating development and operational costs is integral to the justification of new systems development. Methods are needed for creating such estimates before object technology can be fully accepted.

As part of our work at AMS, we have tried to learn from the experience of others how to estimate the time and cost of developing object-oriented systems. You may laugh, but our best summation is to estimate as you would using traditional techniques, then figure that using object-oriented technology will take less. Unfortunately, this hardly supports a major paradigm shift. Before they commit to major changes, managers want to see quantified estimates of system development cost and effort. They need something more specific than "it will take less time." (From the additional perspective of a consulting firm attempting to bid system development jobs, I also need something more to tell a potential client than "it will be better," particularly if I want to be paid for my consultation!)

Training and education

It almost goes without saying that courses, workshops, and other information to educate analysts, designers, programmers, and managers are required to ensure the successful introduction of the lifecycle, methods, techniques, tools, and languages of object technology.

There seem to be two main suggestions regarding training and education in object technology: the "forget everything you know" approach and the "objects build on what you know" approach. I am not sure these divergent viewpoints will ever be reconciled, but I submit that it is difficult for most people to forget everything they know.

To the extent that object technology presents a major paradigm shift, we have to help people integrate these new ideas into their existing mental models of systems. This is a significantly more difficult task than simply teaching someone a new programming language. As an industry, we haven't even had such great success at that. The people teaching "Object Technology for COBOL Developers" will need more than a week to interact with their students if the message is going to stick.

READY FOR PRIME TIME?

The $64 question is whether object orientation is ready for prime time, i.e., developing large-scale strategic business systems. At the current rate of change in technology, the tools and languages should be ready in about 12–18 months. At the rate of change of management processes, it may be a lot longer before the tools and languages can be effectively applied in the IS environment (I am not counting "COBOL programs written in Smalltalk" as effective use).

We have a chicken/egg problem working here: we can't get to large application systems until we have a certain infrastructure available (related to lifecycle, methods, tools, languages, and management) providing a foundation for a successful project, and we can't get that until we try some things to see what works. Obviously, some organizations are taking a giant step (perhaps a "leap of faith") and moving from small-scale testing of concepts to large-scale implementation. Since most large organizations are averse to risk, however, they will need to see significant activity in the areas I have discussed. (What we need is an iterative prototyping approach to the development of object-oriented infrastructure.)

While we can't have all the answers before we start, we should be working on them and willing to talk about the results. Strategic business systems deal with some hard problems. That's where object technology should be able to give our industry the greatest payoff. And if it does, there will be more than enough benefits to go around. ≡≡

References

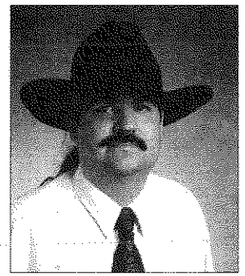
1. Wirfs-Brock, R. The phases of an object-oriented application, THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY 1(5), 1992.

Acknowledgments

The author would like to thank Milt Hess, Andy Baer, and Fred Forman for their comments on an earlier draft of this article.

Dr. Grochow is a Vice President at American Management Systems, Inc., an internationally recognized information systems development firm. As senior member of the Corporate Technology Group, he supervises the introduction of new technologies into the firm's business practices. He has been a consultant on object technology to IBM and is on the Board of Directors of Knowledge Systems Corporation.

What TQM means for OT



Robert Howard

As we accept the challenge of improving our software development efforts using object-oriented technology (OT), we should not neglect the wisdom earned by sources outside the familiar world of computing. The popular concepts embodied in total quality management (TQM) comprise at least one set of guidelines worthy of consideration.

This article outlines the ideas in TQM and suggests ways we can apply them when reorganizing our computing resources around OT. One recurring theme is that object orientation (O-O) allows opportunities for people besides core developers and other project members to get involved in the software development process. The key conclusion is that the move to OT should ideally be part of a larger effort to institute ever-increasing improvement of the overall software development process.

An orientation toward quality has helped many businesses improve their level of customer satisfaction. This often leads to bottom-line improvements such as higher regard for products, improved customer loyalty, and lowered maintenance costs. Computer software has a customer satisfaction problem: standard software product disclaimers usually state that the product is not necessarily useful for any stated or implied purpose. Also, the market's perception is that early versions of new software products are going to be buggy and major upgrades generally late.

Don't count on consumers accepting today's standards of quality for too long. Aggressive companies are already trying to improve upon current perceptions of software and thus take the high road to ultimate success.

Object-oriented technology is for many of us a way to win the constant battle against bugs and backlogs. The hope is to make software development more of an engineering discipline with better product quality, better scheduling, and less job burnout. We should note, however, that many organizations have made improvements in these areas by applying total quality management principles to the development process. I suggest we study their success while we draw up our own plans for implementing OT.

What is TQM? Answers vary, but here are the main points:

- Get it right the first time.
- Meet or exceed customer expectations.
- Commit to continuously improving quality.

These are nice goals we all can agree on, but how do companies actually go about becoming quality-oriented? Here are some guidelines that have worked in many organizations.

ENSURE SENIOR MANAGEMENT COMMITMENT

As readers of this newsletter know, object-oriented technology changes the software development product cycle. This can disrupt organizations. Senior management must back the changes to keep everyone in line with the new program. This is not to be taken lightly. Proceeding with implementation of OT without senior management commitment is a serious step that is highly unlikely to succeed.

ESTABLISH A QUALITY TEAM

The first major step is the formation of a team for quality. This team will take a step back from the problems at hand and evaluate and hopefully improve the organization's software development process. To keep the discussions focused on practical rather than theoretical matters, try to quickly focus on specific

> ***"Aggressive companies are already trying to improve upon current perceptions of software and thus take the high road to ultimate success."***

questions as soon as possible. Assign subteams of one to three people to study the most important topics. Distribute the tasks evenly over time and personnel. If everyone tries to be involved with all issues, they won't have time to make their own contributions to the important projects to which they are assigned. As you will see, there is a lot of work to be done, so plan to expand the scope and membership of the committee over time as new talent reveals itself in your company. The team will concentrate on process-oriented functions that affect the productivity of all

hotline on OBJECT-ORIENTED technology

VOL. 3, NO. 12 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES OCT. 1992

Return on investment: development environments for the lifecycle (part 3 of a series)

Sam Adams

Over 75% of the Fortune 100 have object technology (OT) projects underway; their question is not "*Do* we go object-oriented?" but "*How* do we maximize the benefits of object technology and manage the risks?" This article is the third in a series concerned with meeting the challenges of enterprise-wide computing using object technology.

In previous articles, we have defined the requirements for maximizing return on investment in object technology by creating and managing reusable software assets. We also introduced KSC's lifecycle methodology for the development of object-oriented business systems from requirements to code and back. This article will focus on tool requirements for the deployment of object technology on an enterprise scale.

OOP AND ROI

KSC's experiences have shown that maximizing return on investment (ROI) in OT requires that software be treated as a corporate asset appreciable through investment in its quality and reusability. The most valuable software assets of any organization will be objects that capture the essential nature of their business domain; their foundation will be high-quality design information, not just code. Pervasive reuse of these software assets must become the norm but cannot succeed on a large scale without the existence and proper management of large libraries of software components that are reuseful as well as reusable. For two years now, KSC has funded a major internal research and development effort aimed at defining a methodology and environment for object-oriented software development to help meet these organizations' goals.

REQUIREMENTS FROM THE INDUSTRY

In a previous issue of HOTLINE, Norman Plant presented the findings of the Object Interest Group (OIG), a UK consortium comprising 14 of Britain's largest manufacturing and financial corporations, two government departments, and one university (*Hotline* 3[5]). In evaluating OT the group reached this conclusion:

> Object orientation is potentially one of the most powerful technologies ever to become available to the IT industry and its users. As such it demands high-calibre management. It is not a panacea but a high-powered tool—dangerous if misused but capable of great things.

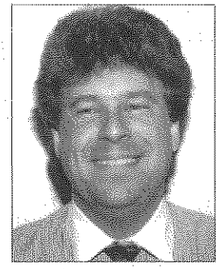
Regarding requirements for development environments, OIG said:

> Given our emphasis on reuse, a development environment will be needed that is rich in class libraries of reusable code, and provides tools to support fast prototyping and intelligent browsing. If people cannot browse easily, reuse will be harder than reinvention and reuse simply will not occur.

continued on page 7

IN THIS ISSUE

FROM THE EDITOR



Our feature article this month is by **Mr. Sam Adams** of Knowledge Systems Corporation, who brings us installment three in his four-part series on getting return on your object technology investment. We have talked before about return on investment (ROI) and software assets, but Mr. Adams phrases the concept in a most enlightening way: "Maximizing ROI from object technology requires that software be treated as a corporate asset that can appreciate through investment in its quality and reusability." This turn of phrase argues equally well for TQM in a manufacturing process that turns out hard goods. The investment is in a process that delivers quality products and is repeatable. The benefit derives from the high-quality reusable and reuseful components the process delivers.

Most discussions of reuse address code-level components. Mr. Adams reminds us that the largest reuse gain comes from design reuse, not just code or component reuse. With the design is captured the semantics, requirements, test suites, and the code for (possibly) multiple language implementations. The semiconductor industry wins in spades from design reuse. Have you ever examined an enlarged IC chip design? Regions on these designs can be clearly identified by the lay person much as any one of us who might not be a botanist could recognize different kinds of flowers in a picture of a neatly blocked garden. Back to the chip, the lay viewer can detect similar regions across different chip types—like an on-board cache region on a CPU chip that looks in pattern exactly like the storage regions on a RAM chip. Large portions of the buffer design were reused—at a significant savings in design cost.

With this perspective in mind, give thought to development environments, languages, design methods, and CASE tools intended to support object software. Most development environments support code (component) reuse. Button, collection, slider, window, bitmap, and canvas classes either ship with the environment or can be purchased from third parties supporting language products inadequate in their raw form for commercial use because they lack such components. The better development environments include code and class-structure browsers that facilitate component reuse.

Design methods and CASE tools for object technology are another matter entirely! Many of today's object methodology practitioners started their careers in structured methods, and propose to adapt their analysis and design methods to objects. Some pundits have substantially rethought the concepts and principles underlying a development method and are taking advantage of appropriate technique and method components from traditional approaches in their new methods. Traditional methods and techniques for design harken from an era of custom construction. Components were not even a dream. Integrated circuits had not yet been commercialized. If I didn't enjoy vintage vacuum tube audio equipment from the same era, I might be so rash as to reference the dark ages, but, alas, my cave is too comfortable for such remarks! As traditional methods are kneaded into something resembling object-oriented development, the archaic value system surrounding custom software design comes along for the ride. Methods that propose to assemble (instead of design) a system would be more appropriate for the economic and political objectives we have in mind for object technology.

As long as the job of our designers and programmers (as reinforced by commercial methods and training) is invention and custom construction, Mr. Adams ROI will not appear. The focus must shift from "my job is to invent it here" to "my job is to translate the problem into a sound solution that can be assembled from parts with minimal invention." And the focus of our corporate reuse efforts must be the management of design components backed up by quality, proven parts. To paraphrase Mr. Adams, code is too poor a medium to communicate business meaning. Reengineering from code can (at best) only extract how the program works, not why the business works that way (if it really does)! We must manage and problem solve at the level of meaning, and construct from parts that are inextricably bound to that meaning.

Mr. Sesha Pratap of Centerline Software picks up the theme of software reuse and discusses what is necessary to cause developers to adopt reuse practices that enact the corporate commitment. The primary objective of training developers in object technology, Mr. Pratap emphasizes, is to educate them in component reuse and cause them to practice it. Corporate commitment notwithstanding, without re-education, clearly set objectives, and strong incentives to steer conventionally schooled developers I am convinced that component reuse will not rise above what Dr. Adele Goldberg calls arbitrary reuse—some reuse happens, mostly at the individual level, and the financial benefit to the organization is limited and tough to measure.

This issue contains the first installment of the Object Methods column, authored by **Ms. Patti Dock** of Orgware, which will address different issues surrounding object development disciplines. In this month's column, she discusses the approach taken by Alcatel Network Systems in evaluating object-oriented methods from Booch, Coad/Yourdon, Edwards (à la Martin/Odell), Graham, Rumbaugh, Schlaer/Mellor, Wasserman/Pircher and Wirfs-Brock for corporate use. For further reading on the subject, we recommend AN EVALUATION OF OBJECT-ORIENTED ANALYSIS AND DESIGN METHODOLOGIES, wherein you will find ANS results discussed in detail, and JOOP FOCUS ON ANALYSIS & DESIGN, which contains a five-way methods evaluation from the Hewlett-Packard Information Management Lab that considers HOOD, Buhr, Booch, Rumbaugh and Wirfs-Brock, both published by SIGS Publications.

While the conclusions of such studies are valuable in and of themselves, Ms. Dock reminds us that the "right" conclusions may vary depending on your organization, its objectives, the experience and background of your team, and the kinds of projects being worked. The greatest value is the reusable

ing the benefits of object-oriented programming will require time and effort, as you must shift to a new style of thinking and move beyond the current structured methodologies.

The language of objects, Bill Wright, **D G REVIEW**, 6/92

DATABASES

. . . Object databases make sense to me. With a traditional database, you must try to represent everything with one of the predefined data types such as numeric, fixed-length string or date. The operations that can be performed on these types are largely predefined. . . . However, most traditional databases are very weak in the ways they can manipulate BLOBS [binary large objects]. Object databases give you even more storage flexibility and efficiency than standard BLOBs, and they let you manipulate these types of data almost as easily as you manipulate numbers and fixed-length alphanumeric fields. . . .

Objective viewpoint, Brad Clements, **D G REVIEW**, 6/92

. . . However, recent interest in object-oriented design and implementation environments is revolutionizing the development of coupled KB/DB [Knowledge Base/Database] systems. Object-oriented concepts such as class and object structures, inheritance, and data encapsulation make separation and coupling knowledge and facts smooth and natural. Combining an object-oriented approach with a hierarchical structure further aids the KB/DB coupling process by helping to identify search and inference patterns that can then be encapsulated into object attributes and methods. . . .

Object-oriented methodology for knowledge base/database coupling, Kunihiko Higa, Mike Morrison, Joline Morrison & Olivia R. Liu Sheng, **COMMUNICATIONS OF THE ACM**, 6/92

IMPLEMENTATION

. . . The tasks in an object-oriented effort are different. New tasks are required to identify, characterize and document objects. These tasks focus on identifying objects and the interactions required of these objects to provide a system that meets stated requirements. Object-oriented efforts, like other development approaches, need requirements and design specifications. Yet these documents localize around objects, and not functions or data. In addition, these specifications clearly delineate which components are reused from an inhouse reusability library and which are developed from scratch to support the application at hand. Tasks associated with the construction of structure charts, data flow diagrams and other function- or data-oriented models are neither appropriate nor useful in an object-oriented effort. Such tasks are obsolete and replaced with modeling approaches more in concert with object-oriented development. . . .

Designing the object-oriented way, Ron Schultz, **OPEN SYSTEMS TODAY**, 7/20/92

. . . One of the most important lessons we learned in our group was that nonreusable code is easy to write in any language. . . Similarly, we learned that reduced maintenance can be realized only when you follow good design practices. . . We also learned that writing poorly performing object-oriented software is easy. . . .

Practical object-oriented programming: 38 guidelines for making OOP work, Bill Hunt, **EDN**, 7/6/92

. . . A key idea of the OOC [object oriented computing] paradigm is that computer systems should simulate the way enterprises and real systems operate. This is in stark contrast to past approaches, which emphasized the decomposition of real systems into procedural specifications that could be easily programmed. The resulting implementations frequently distorted the operations of the enterprise, and were very difficult to integrate and maintain. . . .

Developing CAD: CAD 2001—beyond the event horizon, William V. Weiss and Victor C. von Buchstab, **DESIGN ENGINEERING**, 6/92

THE BUSINESS OF OBJECTS

. . . Perhaps NeXT's most useful accomplishment has been to commercialize object-oriented technology, a concept that has been around since the late 1960s. Once NeXT finally got up and running, other computer firms entered an object race. . . .

The NeXT wave on Wall Street, Jenna Michaels, **WALL STREET & TECHNOLOGY**, 6/92

F.Y.I.

Excerpts from leading industry publications on aspects of object technology

MULTIMEDIA

. . . [according to Marc Stiefler, vice president of development at Xanadu] "The computer industry is just now implementing piecemeal—as computing power increases and market needs demand—the integrated group of elements conceived in . . . earlier research: windows, icons, menus, multiple processes, object-oriented programming and hyperlinks." The real changes will hit in the next year or two. . . .The benefits will be widespread. Searching and reading through hundreds of thousands of pages will become as fast and easy as leafing through a few chapters of a how-to book on cooking or boating. Not only will these electronic books contain text, they will include virtually any kind of information that can be stored on a diskette—audio, visual, graphical or animated. This concept, hypermedia, is simply an extension of the hypertext method that incorporates other media as well as text. . . .

Hypertext: a new world for the document, Bernard C. Cole, **ELECTRONIC ENGINEERING TIMES**, 7/20/92

. . . Object-oriented 3-D offers almost total control of the final image. When each letter is processed as a 3-D object, the title can be viewed from any angle, camera focal length or viewing distance. You can display it from any point of view. As though behind a camera, you can truck in toward the title, truck out from it, dolly right, tilt left. Object-oriented files are also the basic ingredients of animation. By importing a file into a compatible animation program, you can render and compile an animated journey. Instead of seeing a static title, the viewer can experience a flight into, over and around the 3-D credits. . . .

Desktop video: Titles in 3-D, Michael DiSpezio, **VIDEOMAKER**, 6/92

CASE STUDIES

. . . The coupling of CASE technologies with the object-oriented paradigm is starting to lead to a significant expansion in reusability at all levels. Whereas ordinary procedural languages such as Fortran rarely produce more than 15 percent reusable code, with object-oriented languages such as Smalltalk and Objective-C, reusable code often exceeds 50 percent, owing to the concept of inheritance embedded in the language. Although object-oriented analysis and design techniques are not as advanced as the languages, they are advancing rapidly. . . .

CASE's missing elements, Capers Jones, **IEEE SPECTRUM**, 6/92

. . . While object-oriented design has captured tremendous attention in the past few years, associated methodologies are still evolving. . . .Uncertainty surrounding which methodology to use can make it difficult on the software engineers striving to create supporting CASE tools. . . .

Tool vendors "CASE" the industry for object-oriented designers, Amy Bermar, **EDN NEWS**, 6/8/92

LANGUAGES

. . . In a poll of C++ and Smalltalk users, ParcPlace Systems, Inc.'s. Objectworks/Smalltalk compiler and development environment distinguished itself with the top overall score of 79 [out of a possible 100]. Its competitors finished close behind: Microsoft Corp.'s C/C++ scored 77, Digitalk, Inc.'s Smalltalk/V earned a 76, and Borland International, Inc.'s C++ scored 75. The results demonstrated some of the relative strengths of these two approaches to OOP. In general, users gave the two Smalltalk products higher satisfaction ratings in areas such as support for inheritance and ease of maintaining applications. The C++ environments scored noticeably higher in speed of applications. . . More than 70% of the users surveyed indicated that using OOP has increased their productivity.

C++, Smalltalk vie for object-oriented favor, Derek Slater, **COMPUTERWORLD**, 7/20/92

. . . Deciding which language to use may not be as important a decision as is the choice to try a new paradigm for programming. All of the languages discussed here provide the basic tools necessary for OOP. The challenge will be to learn to make effective use of the growing number of class libraries. Reappart: the process. The Alcatel approach can be used as a framework for your own methods' evaluation after tuning the criteria and weightings to your organization's needs.

On a subject near and dear to methods, I remain concerned about GE Advanced Concepts Center and OMTool. While their people are admirably enthusiastic, OMTool is not on par with commercial CASE offerings today. Experienced professionals who have seen the product demonstrated report that its release is seriously premature, suggesting that OMTool is unready for public consumption to the degree one would expect from a product delivered in a shrink-wrapped box to conference audiences. When you stop by the GE booth at OOPSLA, enjoy the virtual reality scene, but keep in the forefront of critical thought the real-world modeling problems you have to solve back at work. When a large, well-resourced vendor is building tooling for a method as widely known as Rumbaugh's, is it unreasonable for the market to demand that they "bring good things to life"?

Mr. Bill Hunt from Hewlett-Packard writes about selecting an object-oriented programming language. His description of the unfortunate way in which many organizations select an OOPL reminds me of lemmings to the sea—without the benefit of population control. With each new technology, the same lemmings will try again, and again, and again. . . The bottom line here is don't select a language or environment simply because the learning curve looks shallow. Languages that are most like the traditional programming language you are using today are the wrong choice if that is your primary reason for selection. Today's development problems arise from today's solutions. The processes we use and environments in which we develop shape our solutions. Recall the sage observation by Ludwig Wittgenstein that language controls what and how we think. Originally pertaining to natural language, this observation is equally valid for programming languages. All language is a medium for concept expression and communication between a sender and a receiver—even when the message is sent to self!

The point is: beware the solution that requires no change. Language vendors take note: the right answer for your customer's business is *not* "It's a better C." As Mr. Hunt emphasizes, the claim to fame of hybrid languages' is their ability to mix object and traditional programming constructs. This has strong appeal for those wanting the appearance of object technology without making a positive commitment. As in much of life, we get back in proportion to what we put in. A hesitant commitment to learning will produce little learning. A play-it-safe, language-only approach also negates the principle benefits object technology offers the business, Mr. Hunt observes, although it may lighten the programmer's work load when coding up a window for display on the graphical user interface. The most important reason for using an OOPL is to produce a higher-quality product, not simply to make programming easier while we keep on doing things the old way.

Dr. Mary Loomis of Versant Object Technology writes about object database management systems (ODBMS), who's using them, and why. Within information technology (IT) organizations, ODBMS vendors are still working to prove their mettle. Today, Fortune 500 companies run as much as a year ahead of middle-sized businesses in exploring new information technology due to their dependence on technology and available resources. It is exactly these businesses that work with instance and concurrency volumes beyond the proven limits of any ODBMS product available today: multi-gigabyte (and terabyte) persistent-storage demands and 1,000-40,000 online user environments. These are the very situations that so desperately require the semantic preservation that Mr. Adams and our frequent author Mr. Norman Plant discuss, and thus that most need the business benefits of object technology and an ODBMS. While the market for structure servers is real and substantial, IT needs large-scale, semantics-sharing servers. Watch this technology and evaluate the products (and vendors' claims) with a critical eye. As the field matures, ODBMS will be a powerful technology for IT.

There is a moral in here somewhere: Look beyond today's solutions to get beyond today's problems, and look beyond today's problems to escape today's solutions.

hotline on OBJECT-ORIENTED *technology*

Robert Shelton, Editor

SIGS ADVISORY BOARD
Tom Atwood, Object Design
Grady Booch, Rational
George Bosworth, Digitalk
Brad Cox, Information Age Consulting
Chuck Duff, The Whitewater Group
Adele Goldberg, ParcPlace Systems
R. Jordan Kreindler, General Electric
Meilir Page-Jones, Wayland Systems
Tom Love, OrgWare, Inc.
Bertrand Meyer, Interactive Software Engineering
Sesha Pratap, CenterLine Software
P. Michael Seashols, Versant Object Technology
Bjarne Stroustrup, AT&T Bell Labs
Dave Thomas, Object Technology International

HOTLINE EDITORIAL BOARD
Jim Anderson, Digitalk, Inc.
Larry Constantine, Consultant
Mary E.S. Loomis, Versant Object Technology
Reed Phillips, Knowledge Systems Corp.
Trygve Reenskaug, Taskon A/S
Zack Urlocker, Borland International
Steven Weiss, Wayland Systems

SIGS Publications, Inc.
Richard P. Friedman, Founder & Group Publisher

ART/PRODUCTION
Kristina Joukhadar, Managing Editor
Susan Culligan, Pilgrim Road, Ltd., Creative Direction
Elizabeth A. Upp, Production Editor
Jennifer Englander, Art/Production Coordinator

CIRCULATION
Diane Badway, Circulation Business Manager
Ken Mercado, Fulfillment Manager
Vicki Monck, Circulation Assistant
John Schreiber, Circulation Assistant

MARKETING
Lorna Lyle, Promotions Manager—Conferences
Sarah Hamilton, Promotions Manager—Publications
Caren Polner, Promotions Graphic Artist

Administration
David Chatterpaul, Bookkeeper
Ossama Tomoum, Business Manager
Claire Johnston, Conference Manager
Cindy Roppel, Technical Program Manager
Helen Newling, Administrative Assistant
Margherita R. Monck, General Manager

Jane M. Grau, Contributing Editor

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012, (212)274-0640.

Mailed First Class. Subscription rate — one year (12 issues) $249, Foreign and Canada $279. Single copy $25.

POSTMASTER: Send address changes & subscription orders to HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville, NJ 07834.

Submit editorial correspondence to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123 voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of HOTLINE ON OBJECT-ORIENTED TECHNOLOGY, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, THE X JOURNAL, C++ REPORT, THE SMALLTALK REPORT, and THE INTERNATIONAL OOP DIRECTORY.

Selecting the right object-oriented method

Patti Dock

Alcatel Network Systems Inc. (ANS), a multinational telecommunication company, is integrating object-oriented methods into its mainstream development activities. ANS was looking for a common sense approach to quantifying the decision-making process. Pursuant to that goal, John Cribbs, Colleen Roe, and Suzanne Moon, all members of the Advanced Projects Group, spent the past year evaluating different design methodologies and issued a 75-page report, *An Evaluation of Object-Oriented Analysis and Design Methodologies*, published by SIGS Publications.

Since reviewing this report, I have had several opportunities to discuss the experience with the authors. In searching for a process that would enable them to stand behind their recommendation, they took a fairly straightforward approach:

- Determine what is important to the organization.
- Specify the evaluation criteria.
- Apply a filter for selection of included methodologies.
- Quantify the evaluation.
- Formulate a recommendation.

Their report is the most in-depth, detailed, point-by-point comparison of current object-oriented methodologies I have seen, covering Booch, Coad/Yourdon, Edwards (including Martin and Odell), Graham, Rumbaugh, Shlaer/Mellor, Wasserman/Pircher, and Wirfs-Brock.

If your company is in the process of evaluating object-oriented methodologies, the ANS experience will save you enormous time. It doesn't matter whether or not your criteria precisely match theirs or if you agree with the weights they used for their rating system, or even if you agree with all of their assessments.

This article combines a brief explanation of ANS's five steps with some thoughts on how to apply them in your own corporation.

DETERMINE WHAT IS IMPORTANT TO YOUR ORGANIZATION

Before considering migrating to a new technology, it is always important to understand your organization's cultural and business requirements.

ANS started the process by categorizing their requirements. In their report, John, Colleen, and Suzanne stress the importance of understanding what constitutes each category and the relative importance of each. They admit that a different staff working in a different problem domain might assign different weights and therefore reach significantly different conclusions.

Whether the method uses an object model and event trace, an object schema and object flow diagram, or a class diagram and timing diagram, all the methods describe the systems from several different perspectives or models. The first step in defining a common basis for comparison was to agree on a common model for evaluation. Only after ANS decided to map each method to a static-logical, dynamic-logical, and physical model were they able to list each individual criterion and determine the relative evaluation weights.

The static logical model captures information regarding the static structure of classes and objects and their relationships. It addresses how well each method defines the class, its relationships, and attributes: Does it capture inheritance? Does it limit the visibility? Are pre- and postconditions explicitly listed? What about version control of individual objects?

The dynamic logical model captures the system's non-static characteristics. A system's time-dependent characteristics such as object lifecycles, events processing, state transitions, flow of control, and timing constraints are represented by data flow diagrams, pseudocode, extended class diagrams, and algorithm statements. Other important issues such as persistency and concurrency are addressed in the criteria.

Transforming the static and dynamic models into a physical model requires information concerning task partitioning, process mapping, task dependencies, and synchronization.

ANS decided to independently evaluate the quality and understandability of the notation. They believe that the usefulness of even the best modeling concepts directly correlates to the ease with which the model communicates information.

ANS found they still had a group of concerns that were not included in any of the previous categories. Among these concerns, or intangible criteria, were:

- Is C++ (at the detailed design level) supported?
- Is there a smooth transition between analysis and design?
- Is CASE support available ?
- How did existing SA/SD environment map to the new methods?

I believe most companies could use the ANS paper as a framework for their evaluation, mapping most object-oriented de-

Partnerships & Acquisitions

Servio UK in London, opened and will provide service to customers of Servio UK and Servio's existing European distributor network. In a separate announcement, Servio has been named a Hewlett-Packard Value-Added Business Partner.

Phil Sheridan was appointed to the newly created position of Vice President of International Operations at **Servio Corporation**. Sheridan comes to Servio from Boole & Babbage Corp., where he served as director of marketing for the company's on-line teleprocessing products.

NeXT Computer appointed Bernhard Woebker as its new vice president of Europe. NeXT's European executives are now located in NeXT's Munich and Paris offices. Woebker will be stationed in Munich along with Herbert Schwab, recently appointed NeXT's head of finance and administration for Europe. In Paris will be Paul Vias, who returns to his position as technical director for Europe after serving as acting VP of Europe since early February, and Randall Sosnick, responsible for legal and business affairs.

Object Design Inc. and NeXT Computer Inc. signed an agreement to jointly develop object-oriented storage and database technology with the intention of creating an industry standard. Under this agreement, they will port ObjectStore 2.0 to NeXTSTEP Release 3.0 by the end of 1992 to be marketed and distributed by Object Design. NeXT will incorporate other jointly developed products into future versions of NeXTSTEP and. Object Design will license these to other vendors.

Ron Lang has joined **Rational** as vice president of marketing. He will manage the marketing effort of Rational's object-oriented products division in addition to the company's corporate marketing functions. Most recently Lang was director of software product marketing at NeXT Computer, Inc.

Michael Sayer has been named vice president of international sales and marketing for **Lucid, Inc.** He will establish European headquarters and manage Lucid's European operations. Sayer has been with Lucid for six years and was most recently general manager of Lucid's Lisp Division. Lucid has also announced distribution agreements with the following European companies: **ELSA Software**, France; **Mesarteam S.p.a.**, Italy; **ENEA DATA AB**, Scandinavia; **Engineering Software Ltd.**, U.K.; and **C.S.E.** Austria, Germany, Switzerland.

Chicago-based **O'Connor & Associates** formed a new division, **Black Diamond Technologies**, to market trading systems software products for NeXTSTEP and other UNIX-based workstations. At the same time, Black Diamond Technologies said it has signed an exclusive two-year agreement with Lotus Development Corporation to distribute Lotus Realtime for NeXTSTEP.

Object Design Inc. announced the Object Design Partner Program for ObjectStore. Among those companies already signed on as members of this Program are **Borland International; CenterLine Software, Hewlett-Packard Company, fluent Inc., Lucid Inc., Oberon Software Inc., ParcPlace Systems, Progress Software Corporation, Spatial Technology, STEP Tools Inc.**, and **Visual Edge Inc.** In addition, public domain class libraries will be part of the Partner Program tool catalog. Object Design is actively seeking new members for the Partner Program.

Object Design and **Progress Software Corporation** announced the signing of a strategic development and marketing agreement under which the two companies will integrate the PROGRESS 4GL and ObjectStore ODBMS. Development of a joint product, codenamed "Object Access Project," has already begun. The first product is scheduled to ship in the first quarter of 1993.

Lucid Inc. and **Artificial Intelligence Technologies Inc. (AIT)** have announced a partnership to bring new Lisp-based tools and applications to market. Under this agreement, Lucid will market and support AIT's standards-based products. The two companies agreed on a plan to release the first product, AIT's Motif tool kit, during the third quarter of 1992.

Oberon Software Inc. opened its western regional office in Palo Alto, CA. William Doerlich is Western Region Manager.

Pencom Software is launching a training and consulting team to serve both developers and end user organizations equipped with workstations from **NeXT Computer.** The new training services will be performed in some cases with NeXTedge, NeXT's service and support group, which Pencom will assist with post-sales support and possible other activities to be announced at a later date. Pencom also plans to initiate NeXT-specific training courses in various regions across the country.

FrostByte Software announced a Value Added Reseller agreement with **NeXT Computer Inc.** to sell NeXT workstations to create custom applications for organizations whose needs have grown beyond personal computing alternatives or who are preparing to downsize from mainframe environments. NeXT workstations running Oracle's RDBMS over an ethernet LAN will be the foundation for large client/server enterprise solutions.

Island Systems

Island Systems has announced additional compiler and graphics support for *object*-Menu, a class library designed to enable graphics developer to quickly integrate a state-of-the art Graphical User Interface environment into any C++, DOS-based application. In addition to support of Borland C++ and Microsoft C++, *object*-Menu will also be available for use with Metaware C++ and Watcom C++ compilers for 32-bit protected mode support. *object*-Menu is also expanding its graphics library support to include MetaWindow (Metagraphics Corp.), Genus GX graphics (Genus Microprogramming Inc.), and HALO graphics (Media Cybernetics). Support for Watcom C++ is pending release of the Watcom product.

7 Mountain Rd., Burlington, MA 01803 617.273.0421

Template Software, Inc.

Template Software is shipping release 4.0 of SNAP, its template-based advanced technology development tools for complex, business-critical applications. This release features the windowed, Menu-driven SNAP Development Environment, which integrates SNAP development tools to ease the creation and modification of SNAP applications, as well as Shared Information Base (SIB), which enables dynamic object sharing among multiple distributed processes, mechanisms to reconfigure or scale applications with little or no code changes required, and links to C++ code.

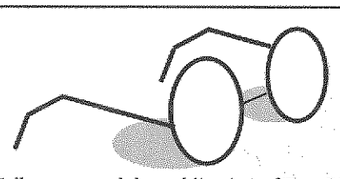
13100 Worldgate Dr., Ste. 340, Herndon, VA 22070-4382 703.318.1000

Cobalt Blue, Inc.

Cobalt Blue has announced the new release for FOR_C++ v1.1, its conversion package offering automated code translation from FORTRAN to C++. Available for MS-DOS and SPARCstations, FOR_C++ translates standard FORTRAN-77 with many MILSPEC and VAX extensions into AT&T's C++. While code is not object-oriented, its translations utilize special C++ objects. FOR_C++ generates complete C++ function prototypes. Function calls are checked for consistent usage during translation and passed either by address or reference in C++. Parameters are translated as C++ constants to help program debugging, with class and structure types being user-defined for greater flexibility

675 Old Roswell Rd., Ste. D-400, Roswell, GA 30076 404.518.1116

BOOK WATCH



Prentice Hall

Prentice Hall announced the publication of THE OI PROGRAMMERS GUIDE by Amber Benson and Gary Aitken. This new guide shows programmers how to build a graphical user interface with the Object Interface (OI) toolkit in the X Window System. OI is a library of class definitions and procedures written in C++, providing a single interface that allows programmers to write programs conforming to both Motif and OPEN LOOK.

Also announced was the publication of OBJECT-ORIENTED PROGRAMMING WITH C++ AND OSF/MOTIF by Douglas A. Young. This book shows programmers how to use C++ and OSF/Motif to design and implement applications featuring interactive graphical user interfaces. It addresses object-oriented programming and design techniques, as well as user interface design methods—emphasizing the thought processes behind each technique—and presents common architectures for object-oriented design.

Another new Prentice Hall release, C++ PROGRAMMING AND FUNDAMENTAL CONCEPTS by Arthur E. Anderson and William J. Heinze, covers the C++ programming language Version 2.1 and its io] stream library. The book offers programmers a clear migration path from C to C++, discussing pre-2.1 versions of the language and how pre-ANSI C and ANSI C differ from C++.

Prentice Hall, 113 Sylvan Avenue, Route 9W, Englewood Cliffs, NJ 07632 201.592.2348

Addison-Wesley

Addison-Wesley Publishing Company and GO Corporation announced the publication of THE GO TECHNICAL LIBRARY, a new series of books, written by experts at GO Corporation, providing an official technical description of the PenPoint operating system. Consisting of seven volumes, these books show experienced programmers how to design, code, compile, and debug PenPoint applications. The GO Technical Library includes: PENPOINT APPLICATION WRITING GUIDE, PENPOINT USER INTERFACE DESIGN REFERENCE, PENPOINT ARCHITECTURAL REFERENCE: VOLUMES I AND II, PENPOINT DEVELOPMENT TOOLS, and PENPOINT APPLICATION PROGRAMMING INTERFACE: VOLUMES I AND II.

Addison-Wesley Publishing Company, 1 Jacob Way, Reading, MA 01867 617.944.3700

> ***Before considering migrating to a new technology, it is always important to understand your organization's cultural and business requirements.***

sign methods into the ANS models with very little customization. The intangible criteria, however, might be significantly different. Some consideratios for determining your intangible criteria include:

- What is the primary purpose of the organization? Are you producing software products or services?
- Is the information organization considered important in the strategic direction of the organization?
- What is the most critical goal of your organization? Is it marketing time, quality, or competitiveness?
- Understand whether your organization's culture is leading-edge or conservative. If you are a conservative company, using a method successfully demonstrated in a similar company may be important. If your company is looking for the competitive advantage, it may be appropriate to use a more revolutionary approach.
- Although ideally one does not preselect a language or development environment, preselection is often the reality. Be sure you recognize historical software development facts. Think about whether the organization has C, COBOL, or Smalltalk experience.
- What is the typical software development lifecycle?
- If the organization has experience in C and has preselected C++, does the methodology support C++? If the organization has a large investment in COBOL, you might transition to object-oriented COBOL, or take a more revolutionary approach with Smalltalk.
- If the organization has a large investment in structured techniques, does the proposed object-oriented methodology allow the use of models such as data flow diagrams to capture class method functionality?

SPECIFY THE EVALUATION CRITERIA IN COMMON TERMS

Next, ANS assigned a value to each category and rated each method according to how well it supports the individual criteria in the categories. They used 1,000 points as the perfect score. ANS's categories and corresponding values were:

Logic Model - Static	250
Logical Model - Dynamic	250
Physical Model	100
Notation	200
Intangibles	200

ANS documented a list of criteria or questions for each category. They assigned a weight to individual questions such that the sum equals the relative weight for the category. Like the selection of the models, this portion of the paper is extremely useful. ANS's initial evaluation had over 70 specific criteria divided among the five different categories. Whether you add, delete, or change the weights, trying to specify each category or question from scratch is a time-consuming task! Here is an example of six of the individual questions that represent 75 of the 250 total points in the static modeling criteria.

		Weight
Class Name	Does the descriptive text name conform to organizational naming conventions?	10
Class Description	Does the text description contain as much and/or as little information as necessary to describe the class?	10
Attribute List	Are the names of the attributes contained in the class	10
Method List	How well does the method document the names/signatures of the methods that define the class's behavior?	10
Abstract Class	Does the method support abstract classes?	5
Unique Identi˚er	How well does the method capture the list of the attributes necessary to uniquely identify an instance?	5
Total points		75

ANS used a rating system and corresponding numeric values to quantify how well each criterion was met:

	Score
Excellent/Complete Support	4
Satisfactory Support	3
Less than Satisfactory Support	2
Unsatisfactory Support	1
Not Supported	0

One of the first questions I asked ANS was "How did you determine the relative weighting of each category?" Their company focus is realtime telephone applications. Having spent five years in the telecommunications business myself, I was surprised that they didn't weight the dynamic logical model more heavily than the static logical model. They indicated that although they had debated the relative weights, the group felt the 5:1 ratio of the logical models relative to the physical model was most appropriate across their entire business spectrum.

Some alternative examples of relative weighing for different systems are:

- A document distribution system might weight the static logical model more heavily due to its ownership and version control dependence.
- A CAD system might weight the physical model most heavily because performance is most often its bottleneck.
- A compiler designer might weight the dynamic logical most heavily because this is where the transformation constructs are addressed.

SELECT THE METHODS TO EVALUATE

Once the criteria specification was complete, ANS selected which methodologies to evaluate. ANS's evaluation criteria stated the method must be described in an English language book, have been used on at least one released commercial product, and be taught by at least one commercial training company.

Given the dynamic nature of the market, John Cribbs indicated that several more methodologies would probably have been included if ANS were to begin the evaluation today. Certainly the publishing industry has been busy this summer and several new object-oriented methodology books are already available.

EVALUATION

Finally, the actual review of each method begins. Here is a portion of one of ANS's charts:

	Booch	Coad	Edwards
Class Name	4 (Excellent)	4 (Excellent)	4 (Excellent)
Class Description	4 (Excellent)	2 (<Satisfactory)	1(Unsatisfactory)
Attribute List	4 (Excellent)	4 (Excellent)	2(<Satisfactory)
Method List	4 (Excellent)	4 (Excellent)	2(<Satisfactory)
Abstract Class	4 (Excellent)	4 (Excellent)	4 (Excellent)
Unique Identifier	1(Unsatisfactory)	1(Unsatisfactory)	1(Unsatisfactory)

Once the individual rating is captured, an evaluation table is compiled by applying the weight of each item in a manner consistent with the item's importance to your organization and summing all the columns according to the formula on the chart.

	Weight	Booch	Coad	Edwards
Class Name	10	4	4	4
Class Description	10	4	2	1
Attribute List	10	4	4	2
Method List	10	4	4	2
Abstract Class	5	4	4	4
Unique Identifier	5	1	1	1
_ weight(i) * score(i)		185	165	115

I caution you not to underestimate the time required to understand the nuances concerning the different modeling techniques, the notation, and the tools supporting it. By definition, each method has at least one book associated with it. Most have tools and some have multiple tools that need to be reviewed. Many have either public or in-house training courses to attend. I suggest you check out at least one reference project for each method.

After the numbers are calculated, a "best" method will be apparent. Be sure to:

- allow time for feedback from your peers;
- provide a mechanism for feedback from your vendors;
- identify any special areas of risk.

You will still need to formulate a migration plan, evaluate the impact on your organization, and estimate the associated costs. After completing these actions, determine the method to recommend, including any modifications necessary for your organization, and present your recommendations.

SUMMARY

The process described in this article should be viewed as a means of making a final selection; it should help to eliminate the biases that are inevitable when making decisions concerning new technologies. Only through a quantitative process can you formulate a defensible recommendation.

I recommend you read the entire 75-page report. Copies can be ordered from SIGS Publications at 212.274.0640. ≡≡

Reference

Cribbs, J., C. Roe, and S. Moon. AN EVALUATION OF OBJECT-ORIENTED ANALYSIS AND DESIGN METHODOLOGIES, SIGS Books, New York, 1992

Patti Dock recently joined OrgWare, Inc. as Vice President. She has been involved in the object-oriented marketplace since 1985 when she joined Stepstone as a technology consultant. Since leaving Stepstone, Patti has worked for Jackson Systems Corporation and General Electric, both companies actively involved in object technology. She currently teaches a course called OBJECTMethods, which compares and contrasts leading object-oriented design methods. She can be reached at 203.270.1242.

Product Announcements

Product Announcements is a service to our readers. It is neither a recommendation nor an endorsement of any product discussed.

Glockenspiel

Glockenspiel released CommonBase, its C++ framework for object-oriented development of database applications. CommonBase provides SQL and ISAM developers with application portability across database platforms, interoperability with other C++ class libraries, a transparent migration path from ISAM to SQL, and independence from database store formats. This new version provides support for two additional platforms: Microsoft's C/C++ 7.0 compiler and Faircom's ctree Plus. CommonBase 1.3 also corrects all known bugs.

39 Lower Dominick St., Dublin 1, Ireland +353.1.733166

UniSQL, Inc.

UniSQL announced UniSQL/X for IBM RISC System/6000 workstations running AIX, providing IBM RS/6000 users with object-oriented data modeling and multimedia data-integration capabilities in an ANSI SQL-compliant environment. UniSQL also announced UniSQL/4GE Application Development Environment, GUI-based application development tools for IBM RS/6000 users.

9390 Research II, Ste. 220, Austin, TX 78759-6544 512.343.7297

Oasys

Oasys introduced Oasys Native SPARC Tools. The Tools include the new Version 1.8.6 release of the optimizing Green Hills Compilers (C, C++, Pascal, and FORTRAN) and the multilanguage, X Windows system-based, MULTI debugger, and have been integrated with Sun's native assembler/linker. Oasys Green Hills Compilers support three intercallable high-level languages: C, switch-selectable ANSI C, and classic K&R C; a production-quality C++ compiler, that is source-code compatible with AT&T cfront 2.1, 2.0, and 1.2, with switch-selectable support for C; ANSI/ISO Level 1 Pascal; and ANSI FORTRAN-77 with DoD MIL-STD 1753 extensions and VAX/VMS extensions. In addition to native SPARC, each language compiler is available in native and cross mode for a wide variety of targets and platforms.

One Cranberry Hill, Lexington, MA 02173 617.862.2002

Metrosoft

Metrosoft announced October shipping of MetroTracks, a commercial multitrack audio software for NeXT workstations, running NeXTSTEP 3.0. A NeXT workstation equipped with MetroTracks and any of several analog or digital audio front ends provides users with a full-featured digital audio workstation capable of recording, mixing, and editing CD-quality sound plus MIDI data. A recording can contain up to 32 virtual tracks in any combination of audio and MIDI and as many as 8 tracks can be mixed simultaneously, depending on the hardware configuration. The MetroTracks architecture is expandable to utilize any future NeXT hardware directions.

740 13th St., Ste. 503, San Diego, CA 92101, 800.851.8665, 619.488.9411

ProActive Software, Inc.

ProActive Software unveiled its Customer Information Resource (CIR) system, a comprehensive family of enterprisewide, client/server applications and tools designed to help companies improve customer satisfaction. ProActive also announced immediate availability of Support Advantage for Microsoft Windows 3.1, the first of six CIR applications, as well as the ProActive Toolset, an object-oriented customization environment. ProActive's CIR system will support multiple SQL relational databases (Sybase, Informix and Oracle), UNIX platforms, and user interfaces (including Windows 3.1,The X-WindowsSystem, and Macintosh). It also features its own workflow engine and Query-by-Example facility. Developed in C++ with a client/server architecture, CIR is built around a Dynamic Application Dictionary and the ProActive Toolset, which enables users to customize their applications without changing source code. Currently available on Sybase, Support Advantage will run on Informix and Oracle database servers on a wide range of UNIX platforms. Support Advantage uses the RDBMSs to access, capture, and track complete up-to-date profiles of customers. Support Advantage also uses Object Linking and Embedding (OLE) technology to link and attach multiple data types.

1043 North Shoreline Blvd., Mountain View, CA 94043 415.691.1500

Microsoft Corporation

The new Microsoft C/C++ Browser Toolkit lets you access and manipulate the contents of the browser database (.BSC) files in Microsoft C/C++ Version 7.0. You can also use the new APIs to develop custom applications that work off the information contained in the database files. Toolkit can be downloaded from Microsoft developer forums on CompuServe or ordered directly from Microsoft.

One Microsoft Way, Redmond, WA 98052-6399 206.882.8080

mentation of the code. The startup cost for a reuse project can also include expenses for hiring additional staff to support reuse and for developing custom tools.

GETTING STARTED WITH SOFTWARE REUSE

The three requirements for implementing software reuse are tools, reusable code, and training. New tools are required because the traditional compiler and debugger toolsets are not adequate. Interactive programming environments are necessary to develop reusable software at the object level. These support object-level prototyping, execution, debugging, and testing, making it possible to develop reusable objects and reuse software at the object level. These environments also provide tools for browsing the structure of objects to determine their dependencies and interfaces.

When beginning a reuse project, start with reusable code in the form of class libraries or an application framework. Purchase as much reusable code as possible, although this approach does depend on the quality of the reusable code. Groups engaged in software reuse should develop criteria for evaluating and testing the quality of purchased software. Class libraries and reusable components may require several releases before they reach acceptable levels for robustness and quality. Training is also vital to help programmers understand how to make the transition to object-oriented programming and software reuse. One of the main objectives of training should be to break old habits that cause programmers to distrust reusable software written by others.

SUCCESS WITH SOFTWARE REUSE

Success with software reuse depends on a few simple rules. First, pick a small project and set realistic expectations for the results expected during the transition to objects and software reuse. Second, make a long-term commitment to improving software reuse instead of a heavy upfront commitment that may have to change. Third, don't expect big rewards until several releases of an application have been built with reusable objects. Fourth, make an investment in tools, reusable code, and training.

Software reuse can be an extremely effective means to reducing software costs, shortening development cycles, and reducing the complexity of creating software. These rewards depend on the level of software reuse and the commitment to making the transition to reuse successful. ≡≡

Sesha Pratap is CEO, President, and cofounder of CenterLine Software, Inc. Prior to launching CenterLine Software, Mr. Pratap specialized in evaluating potential investments in computer and computer-related industries for a venture capital firm. Mr. Pratap also has more than ten years of experience in systems programming and lecturing. In 1990, he was elected a trustee of the Massachusetts Software Council. Mr. Pratap is also a board member of the Object Management Group.

OBJECT DATABASES *continued from page 15*

SUMMARY

Figure 1 classifies the various ODBMSs according to this categorization of customer need.

Please note that this is just one approach to classifying products. I've forced each product into a single category but the majority of ODBMSs are sold successfully in both the persistent storage and database management categories.

My experience with this approach is that some ODBMS customers who start out requiring persistent storage eventually have database management needs. Sometimes this is because a software engineer's success with an object programming language and ODBMS leads to broader adoption of the technology. And broader adoption of the technology typically means using it to support multiuser applications where sharing of objects is mandatory. Some database management customers evolve to needing groupwork functionality, while others start with those needs. Keeping your broader requirements in mind will help ensure that you are satisfied with your product selection. ≡≡

Mary E.S. Loomis is Vice President of Versant Object Technology. She can be reached at 4500 Bohannon Dr. Menlo Park, CA 94025, via phone at 415.329.7500, via fax at 415.325.2380, or via email at mloomis@osc.com.

	Persistent Storage	Database Management
Object	db_VISTA III POET --- with significant groupwork functionality: ObjectStore	Gemstone OpenODB O2 --- with significant groupwork functionality: ITASCA Objectivity/DB ONTOS DB VERSANT
Conventional	DOS and UNIX Files	ORACLE, Sybase, etc.

Figure 1. A classification of ODBMSs.

RETURN ON INVESTMENT *continued from page 1*

Although we agree wholeheartedly with OIG's assessment, we have shown in previous articles that to get return on investment through reuse, organizations must focus on developing software assets that include more information than just code.

The program code itself is a very poor medium of exchange for the wealth of design information developed during a large project. The majority of semantic information about a reusable object or group of objects would be traditionally referred to as analysis and design information; the code has little semantic or reuse value without it. If an organization is to be successful in the development of a reuse-based infrastructure, it is essential that all semantic information be managed in a single consistent model that includes not only code but requirements, design, and testing information as well.

Mr. Plant listed the features of a good O-O development environment, regardless of the actual programming language involved:

- incremental compiler/linking
- change-management support
- single- and multiuser support
- runtime inspectors
- message-passing protocols
- storage management
- dynamic binding
- type-checking facilities
- performance-analysis aids

These features are all essential for the development of O-O programs, but what about requirements, design, user interfaces, and test suites? How can any IS manager be expected to have confidence in the quality and reusability of the software developed when such a small portion of the total process is being addressed by the development environment?

THE TOOLMAKER'S CHALLENGE

The large amount of interwoven information required for business system designs, whether object-oriented or not, must be managed throughout the lifecycle to protect the organization's investment in its reusability and value as a software asset. In addition, version control and configuration management of all relevant information is mandatory if large group design and development is to be managed with confidence. We need multiuser environments supporting all aspects of the lifecycle for the high level of integration required to deliver high-quality software in a large enterprise.

Code generation is insufficient

Many current vendors of analysis and design tools, whether object-oriented or not, rely on a code generation strategy in their attempts to integrate the lifecycle. Even where the amount of code generated from design documents is 80% or more, code and design are out of sync as soon as a programmer modifies the generated code to provide the required functionality. When that happens, the value of both code and design goes down dramatically, especially after several iterations. Many large organizations using these types of tools and strategies have experienced this. Code generation from high-level design information cannot yield complete, executable systems on the scale required by today's business enterprises. But what if the design contains low-level design information as well?

American Management Systems Inc. (AMS) is a large systems integrator with over 25 years of experience in the design, development, and delivery of large-scale government and commercial information systems. Jim Talvitie, a senior principal at AMS, describes the problem this way:

> There is a point when the design becomes so detailed that it is really code with the unfortunate character of being unexecutable.

Design documents, whether textual or graphical, all too often end up this way, with page after page of textual or graphical pseudocode. Programming is still programming, whether done in an executable programming language or in a "design" notation. Designers may spend all their time writing the program, leaving programmers little option but to patch it up until it works or ignore the design and completely rewrite the code.

Automatic code reengineering is insufficient

"So what if the programmers have to 'finish' the code? Even if they completely ignore the design we can use re-engineering tools to extract the design." This is a common claim of tool vendors who use the code generation strategy. But no existing programming language is rich enough to capture all the design intent of the original programmer, whether a human or a code generator. Where are all those wonderfully complex diagrams; the pages and pages of textual description; the requirements and user interface specifications stored in the program code? The fact is, they aren't there. Code is just too poor a medium to hold and preserve this valuable information. The best result we can hope for when parsing program code is to determine how the program works, not why it needs to work that way to meet user requirements.

Required: a common model for lifecycle information

When information is transferred between dissimilar mediums, some of it is always lost in the translation. "Bolting together" a diagram editor with a code generator and a code parser does not provide an appropriate platform for the management of strategic corporate assets, but rather one primed for the loss of valuable design information. Only by providing a single unified model for the integration of all lifecycle information can this "noise" problem be avoided. If the requirements, design, and implementation are provided in the same medium, there is no need for translation and thus no opportunity to lose information.

If this sounds like the well-worn argument for CASE tools and information modeling, keep in mind that the real proof of any idea lies in the delivery of an effective implementation. Traditional CASE tools are seriously compromised by the "data only" or "data + function" perspective. These are the central tenets of traditional design and programming, and their inability to provide a rich and flexible medium for a company's business rules

has long been recognized. Even the most basic definition of an object as an interacting behavioral entity provides a much better foundation for modeling an enterprise.

That is why we base our methodology on these behavioral entities. It is also why the process of developing high-quality designs requires a focus on behavior and interaction. This unified view has already proven highly beneficial both in our work with CRC cards and the methodology presented in earlier articles in this series. The experience also has led us to build lifecycle development environments based on the same principle. But how can a single environment address the entire software lifecycle if it focuses only on high-level design and ignores the development of program code itself?

LANGUAGE-NEUTRAL VS. LANGUAGE-SPECIFIC

For several years, most methodologists in the OOP industry have been advocating approaches "neutral" to the particular programming language chosen to implement the design. While this approach is definitely practical from the standpoint of selling methodology books, it completely ignores the reality of the software lifecycle. Of course, at the earlier stages of a software project, a design can be general enough to be reasonably implemented on a variety of language platforms. But the further toward implementation you go, the more important language features and differences become. Smalltalk and C++, the two most prevalent O-O languages, often produce radically different implementations for the same high-level design. A complete lifecycle development environment cannot ignore this fact. It would be wonderful if we had an environment that would allow us to use any O-O language we like and still provide the integration of all the lifecycle information. Unfortunately, these languages have many more differences than similarities. A good implementation de-sign in a hybrid language like C++ often makes a lousy design in a pure object environment like Smalltalk, and vice versa.

So, while a methodology should be language-neutral at the earlier stages of the lifecycle, it must be language-specific in the later stages. The same is true for any tool or environment that claims to address the requirements of an entire development lifecycle.

MORE REQUIREMENTS

So far we have discussed only the tool needs for the development of O-O software; we have not addressed legacy systems, cooperative processing, or database integration. Although not all organizational applications require these non–O-O components, the environment needs to be able to accommodate and assimilate them smoothly. Since the behavior/interaction approach provides the best way to model these entities as well, the development environment could also support and maintain this information within the same unified model mentioned previously.

MEETING THE TOOLMAKER'S CHALLENGE

Delivering on all the requirements described above is obviously a tremendous challenge. No CASE tool or programming environment available today can even come close. But if OT is to live up to market expectations and large organizations are to receive a profitable return on their investment in this technology, then such an environment must be developed and deployed as soon as possible.

At KSC, we treat these requirements as design specifications for the tools we are building in support of our lifecycle methodology. We recognized over two years ago that a much higher level of tool capability would be required to fully exploit OT in the business computing world.

The next article in this series will describe what we call *Constant Quality Management*, a strategy and process for the development, reuse, and management of software assets. We will show how using integrated testing and constant metric feedback at all levels can enable the entire software lifecycle to be managed with confidence, producing consistent, high-quality results from requirements to code and back. ≡≡

Sam Adams is the Senior Consultant and cofounder of Knowledge Systems Corporation. Since 1984, Mr. Adams has been actively developing object-oriented software systems in Smalltalk and is widely recognized for his expertise. He is codeveloper of the group facilitation technique using CRC cards and has been training computer professionals in object-oriented technology for over six years. Mr. Adams has served on several conference committees and is a frequent speaker and panelist at leading industry conferences. He can be reached by phone at 919.481.4000 or by fax at 919.460.9044.

CALENDAR

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: Robert Shelton, 1850 Union Street, Suite 1584, San Francisco, CA 94123; fax: (415) 928-3036

October 18–22, 1992	October 29–30	Nov. 16–20, 1992	February 1–4 and	April 19–23,1993
OOPSLA	INTEROP	C++ World	February 4–5, 1993	Object Expo
	San Francisco, CA		OOP '93 and C++ World	
Vancouver, BC	Contact: 415.941.3399 or	Meadowlands Hilton, NJ	Munich, Germany	New York, NY
Contact: 407.628.3602	800.INTEROP ext 2502	Contact: 212.274.9135	Contact: 212.274.9135	Contact: 212.274.9135

standard mechanism for applications to communicate with reusable objects.

> *Companies such as GTE have successfully increased reuse in a short period of time by paying commissions and royalties to employees who produce and support reusable software.*

Domain-level reuse consists of reusing software that provides common functionality for a particular domain of software, such as database applications or applications with graphical user interfaces based on Motif or Windows. Domain-level objects usually incorporate the capabilities of many foundation-level objects so that programmers can eliminate the need to write major portions of an application. For example, an editable text window can be used in place of program code built on foundation objects that provide panels, scrolling, and cut-and-paste capabilities.

Application-specific objects or application frameworks allow programmers to maximize the level of reuse for a given program. An application framework consists of a collection of objects that perform functions common to a particular class of application, such as software for financial, manufacturing, CAD/CAM, or telecommunications applications. For example, an application framework for a financial-planning application could provide components for reading and plotting stock information, analyzing historical trends, computing present values, and managing a portfolio. By starting with an application framework, the programmer can focus on writing new code that provides custom functionality unique to the particular application such as creating a financial-planning program for managing overseas stocks.

From an organization's perspective, reuse of software also occurs at three levels: individual, group, and corporate. Individual-level reuse consists of software reused by the same programmer over several generations of a software application as it is continues to be maintained and enhanced. Maintenance activities can be greatly reduced by building modular programs consisting of reusable objects. By using object-oriented techniques and planning for reuse, programmers can preserve the architecture and modularity of their programs as the software is modified. Reuse of software by the individual programmer requires the least commitment and investment by the corporation and therefore is the form of reuse most widely practiced today.

Reuse of software by groups involves the development and maintenance of class libraries shared by a group of developers, usually working on the same project. For group reuse to be effective, the group must have support for the collection, classification, and maintenance of reusable objects. This task can be shared by members of the group but a dedicated individual is required once the library of reusable objects becomes large enough to require continuous maintenance and updating.

Reuse of software at the corporate level requires a formal commitment to software reuse. Most companies form several teams of developers responsible for identifying reusable software among various development efforts, cataloging and classifying reusable objects in a corporate repository or library, maintaining and enhancing the reusable code as corporate requirements change, and assisting programmers in reusing the software. Implementing corporate-level reuse requires a good understanding of the company's software development needs and the areas in which it can share the results of individual development efforts.

BARRIERS TO EFFECTIVE SOFTWARE REUSE

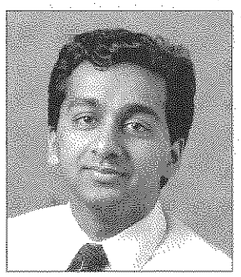
To reuse software effectively, most organizations need to overcome several barriers, the most common of which are technical. Because object-oriented tools and class libraries are relatively new, the first requirement for effective reuse is that tools and libraries assembled from various vendors are compatible and adhere to common standards. For example, in the C++ world, Version 3.0 of AT&T's C++ implementation is emerging as the industry-standard language specification, but programmers are faced with the choice of several differing implementations and several foundation class libraries.

For companies attempting to increase the level of group and corporate reuse, a major technical barrier is the lack of appropriate tools. Commercially available software development tools for configuration management, documentation, and testing do not support object-level software development or software reuse; tools to collect, store, classify, and retrieve components are just becoming available. The easiest solution for many corporations is to develop their own custom tools.

Organizational barriers can hinder the success of reuse programs if the organization does not make a long-term commitment to reusing software. Introducing software reuse to a development project can result in upfront delays because programmers have to learn new tools and programming techniques. Some companies switch course after an initial bad experience with software reuse, which is unfortunate because the payback may not be apparent until after several releases of the software. Organizational history and an unspoken emphasis on measuring success by written lines of code can hinder the commitment to reusing software. This inertia can be overcome with policies that provide recognition and financial rewards for developing and using reusable software. Companies such as GTE have successfully increased reuse in a short period of time by paying commissions and royalties to employees who produce and support reusable software.

For many, the economic barriers to software reuse present the biggest obstacle. Introducing object-oriented techniques and software reuse to a development group can result in high startup costs for training and tools and loss of productivity during the transition. Critical software projects can be delayed because programmers are using new techniques and writing reusable software requires much more effort for the initial design and imple-

Objects and reuse



Sesha Pratap

The concepts of objects and object-oriented programming are already more than two decades old but have been embraced by the mainstream of the software development community only in the last few years. One of the more compelling reasons to adopt object-oriented development techniques is the promise of more effective reuse of software components. This article explores how object-oriented programming promotes software reuse and how software development organizations can adopt tools and techniques to increase software reuse.

WHEN LESS IS MORE: THE REWARDS OF REUSING SOFTWARE

Increasing software reuse is compelling because of the rewards that reuse provides: increased programmer productivity and software quality and reduced complexity. The power of software reuse is based on the simple concept of leveraging programmer expertise and effort. The most fundamental way to reduce the length of the software development process, improve the quality of the software developed, and simplify the software development lifecycle is to reduce the overall amount of code that needs to be designed, developed, and tested. The most effective method for minimizing the development of new code is to maximize the reuse of existing code.

By reusing software, developers are able to reduce the amount of time needed to develop, test, and debug new code. Reusing software usually requires developers to make a greater investment in designing an application and, if the reusable software is internally developed, in maintaining and enhancing existing software. These increased investments in design and maintenance usually mean that productivity gains of reusing software do not become apparent until several generations or versions of an application have been built using reusable objects. Studies by corporations using internally developed class libraries indicate that a software project evolves through four or five versions before the reuse of software produces significant savings in development effort.

Software reuse also helps improve the overall quality of the developed program. Reusable objects can be tested and debugged to provide highly reliable software components, although the most effective means to improve the quality of reused software is to correct the problems identified by repeated use. Using object-oriented design techniques will produce software that is more modular and maintainable. Reusable objects also promote the concept and the practicality of unit testing, thus making testing an integral part of the software-development process.

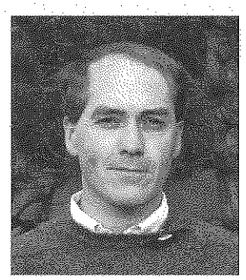
Reusing software can simplify the software-development lifecycle in several ways. First, reusing software reduces the amount of code that needs to be designed, written, tested, and debugged. Second, object-oriented programming and software reuse allow programmers to work at a higher level of abstraction, thus reducing the need to conceive of and work with a software application at the language level. Reusing software also makes it easier to prototype and develop software in a rapid incremental way, allowing programmers to evolve simple solutions to complex programming problems.

WHAT COUNTS AS SOFTWARE REUSE?

Almost all software developed today contains a considerable amount of reused code. Software programs are built upon several layers of system software and application libraries, all of which are reused code. For example, a program that makes calls to the operating system or displays output through a graphical windowing system is reusing code by taking advantage of the functionality provided in the system software. Most programs are also built with functions that are linked from libraries of code. This form of software reuse is prevalent and straightforward but represents only one way to reuse software. The other way is to reuse code written as part of an application over several versions of the application or by a group of similar applications. Object-oriented programming techniques increase both types of software reuse but the biggest benefit of developing with objects is that programs can be built from objects that are easily reused for new versions and new applications.

From an industry perspective, software reuse occurs at three levels: foundation, domain specific, and application specific. Software reuse at the foundation level consists of using industry-wide software components, such as operating-system services and commercially available foundation-class libraries. This represents the broadest form of reuse; the reused software performs activities required by the broadest range of applications. This form of reuse places the greatest demand on the reliability and standardization of the reused software. Most software designed for foundation-level reuse can be accessed through standardized application programming interfaces (APIs) that provide for effective encapsulation of the reused software. The introduction of object request brokers (ORBs) and a common object model will make it much easier to reuse software at the foundation level by providing a

Choosing an object-oriented language



Bill Hunt

Commitment to using object-oriented technology for your next project requires careful planning and thought and is not a decision to be taken lightly.

Before choosing a language, consider the reasons for using an object-oriented approach in the first place. Many new to object-oriented programming assume that to become object oriented all that is necessary is to start writing code in C++. This, of course, is silly because no language can make or break any system by itself and, more importantly, the term object-oriented refers to the architecture of the system, not to the features of the language in use.

One of the major pitfalls encountered when developing an object-oriented system for the first time is the tendency for developers to implement a traditional architecture using an object-oriented language. This is a very easy mistake to make with hybrid languages such as C++ that do not explicitly encourage an object-oriented design. In C++, passing messages and calling procedures both use approximately the same syntax and have almost the same semantics so it is very easy to confuse the two and slip back into the traditional architecture. This does not happen when using the so-called "pure" languages (such as Smalltalk, Actor and Eiffel) because there is no procedure-called syntax at all. Other hybrid languages, such as Objective-C, use a different syntax for passing messages. The developer still has a choice of using a pure object-oriented architecture or a non–object-oriented architecture but cannot easily mix the two.

The ability to mix object-oriented and traditional architectures is very appealing to those who do not want to fully commit to using object-oriented techniques. This is why C++ is so popular. People say "I'll use C++ so that if I don't like this object-oriented stuff I can still use a traditional approach." Languages like Objective-C and Smalltalk require a full commitment to the object-oriented approach.

GOALS OF OOP

Many people new to object-oriented technology don't know why they should be using it. Some say object-oriented technology should be used in order to generate reusable code, while others say that programmer productivity should be the main goal.

In my opinion, the most important reason to use object-oriented technology is to generate a higher-quality end product. This is made possible through strict partitioning of various parts of the system and through the packaging of data and associated procedures together as one unit. When the internals of a system are strictly separated, the code is much simpler and easier to understand. The developers will therefore make fewer mistakes and the result will be a more robust, higher-quality system.

A major side effect of a simpler architecture is higher programmer productivity. This is because developers do not need to spend a large portion of their time understanding the interactions between various modules. Unfortunately, using a language with complex syntax or semantics will cancel the expected productivity gain. Languages with simple syntax and semantics allow developers to implement new capabilities correctly the first time without spending time correcting their mistakes.

Object-oriented systems often have lower maintenance costs because of the simplicity of the architecture. In a good object-oriented design there are fewer interactions between seemingly independent modules in the system. Therefore, a defect can be fixed or an enhancement made without introducing new defects into the system.

In many cases, the overall testing effort can be reduced because of lower defect rates. If the rate is truly lower, fewer defects need be uncovered during the testing phase to reach the same overall quality level. Therefore, either less testing is needed or the same amount of testing can be performed to yield a higher-quality result.

Code reuse is the benefit most often associated with object-oriented programming. It is perhaps the most over-promised and under-delivered benefit of object-oriented technology. Some early reports implied it is *impossible* to write non-reusable code using an OOPL. Actually, it is very easy to write non-reusable code in any language. Sometimes the situation is worse with an object-oriented programming language because of a mistaken belief that the technology will automatically cause the code to be reusable. Unfortunately, the term "reusable" does not mean that reuse is really practical, just as the term "recyclable" does not imply that a product will actually be recycled.

The most important reason that code reuse is not as prevalent as expected is because the true cost of reusing code is much higher than most people realize. For example, in class libraries that have enough functionality to even bother with reuse, there are usually some name or convention conflicts. Some effort must be spent resolving these conflicts (renaming classes, converting data types, etc.) before the code can be reused. This is generally not an insurmountable problem but clearly it is not without some cost.

Often the development team on an object-oriented system can be smaller than for projects using a traditional approach. Because there are fewer details to remember, such as intermodule dependencies, one person can manage a larger portion of the system. Developers are more productive when part of a smaller team: additional team members increase the communication and training load on the rest of the team as well as the chance for miscommunication.

LANGUAGE FEATURES

One important way to evaluate an object-oriented language is to determine how well it supports the object-oriented architecture being developed. Other factors include the availability of software development tools and the existence of usable class libraries. Too often the language is chosen based on less important factors such as comfort level, ease of training, popularity, etc. (this explains the popularity of C++). Unfortunately, less threatening languages are usually a poor choice because it is too easy to slip back into the traditional approach.

Proponents of certain languages will usually find a unique feature in that language and promote it as one required for object-oriented development, implying that it is the only "serious" object-oriented language. In truth, very few features are so important that they are required for object-oriented development.

Essential language features

The following features are absolutely essential for object-oriented development.

Encapsulation Encapsulation is the fundamental reason that object-oriented technology is worth the effort. The ability to declare data and their operations in one module and then restrict access to the data from outside the module allows the programmer to divide the problem into manageable pieces. This is so important that some non-object-oriented languages such as Ada support encapsulation as well.

Polymorphism and dynamic binding Polymorphism and dynamic binding together are critical to the success of object-oriented programming. Polymorphism allows code to be written without specifying the type of its data. Dynamic binding is required for correct execution of polymorphic code because it insulates the sender of a message from the place where that message is received and implemented. Some supposed object-oriented languages don't have dynamic binding at all and some, such as C++, allow it but not as the default. In C++, since static (compile-time) binding is the default, an enormous number of very subtle errors can be introduced unless strict conventions are used.

Inheritance Inheritance allows code to be written just by specifying the differences from existing code. This very effective way to increase the reuse of routines already being used for the project is what I call "internal reuse" and is an important way to measure the potential for future reusibility of the code. Think about it: if the code is already in your system and you can't reuse it, why would it be useful for another project? Unfortunately, inheritance is often overused because it is an implementation convenience that is hard to resist. The problem is that inheritance adds complexity to the system by creating dependency links between a class and the classes from which it inherits. Each time a class is changed, there is a possibility that the change is incompatible with a subclass and that a defect will be introduced into the system inadvertently. Keeping software quality in mind, therefore, inheritance must be used with caution.

It is clear that with enough conventions, macros, rules, etc., C (or even assembly language) can be used to implement an object-oriented architecture, although it is more difficult. However, when considering some of the available object-oriented languages and goals, such as higher quality, higher productivity, lower defect rates, etc., it is important to choose a language that supports these essential features.

Other language features

There are many other language features that can be evaluated when selecting a language for object-oriented software development. Some seemingly important features are actually unimportant or even "anti-features." Others are controversial at best.

Strong vs. weak typing Strong typing allows compilers to do more checking for type mismatches and other trivial errors e.g., sending a message to an object that does not understand it. Proponents of strong typing argue that programmer productivity is increased when the compiler performs additional checks. Some also claim that testing time is reduced because the compiler does not allow certain types of problems to exist in the generated code. Unfortunately, even though additional error checks in the compiler are useful, they cannot detect the subtle semantic errors that are the most common type of defect introduced by experienced developers. Due to the potential for these undetected defects, testing time really cannot be reduced just because a stricter compiler is in use.

The cost of the convenience of extra compiler type checking can be very high. In languages that support only strong typing, it is easy to write code that can handle only one data type. This is fine if the code will not be reused for a different purpose but, if it is desirable to reuse it, the code may need to be modified to handle another data type. Using a strongly typed language certainly does not preclude writing reusable code, but it often requires more effort. One way to increase programmer productivity is to avoid spending this extra effort to design fully general-purpose classes. Not only is it much quicker to design a class for a specific purpose, but predictions about future needs are usually wrong.

In languages that support weak typing, it is natural to write code to support all data types. Therefore, developers can quickly design classes for current needs only, and sometimes the resulting classes are already general enough for later reuse without modification. For example, a collection class originally implemented for the specific purpose of holding a collection of windows is likely to be usable later (without modification) to hold a collection of real numbers.

typically new code or major rewrites of existing code. Customers usually select their object-oriented development methodology and tools, object programming language, and ODBMS products together. They expect to be able to build their applications faster because the combined suite of object products provides the synergy of a unified model from the glass to the disk. Dealing with objects throughout avoids time-consuming and error-prone work to transform between various user-interface, programming-language, and database models.

THESE PRODUCT EVALUATIONS FOCUS ON ARCHITECTURE

Product evaluations in the object database management category tend to focus on the architectures of the various candidate ODBMS products, specifically their ability to support production levels of shareability. Evaluations include investigation of concurrency-control policies, locking techniques, transaction-management capabilities, logging and recovery capabilities, schema evolution, approaches to object identification, etc. Evaluation teams study the candidate products' architectures and considerations like the partitioning of ODBMS functionality between the client and server modules of a product.

ODBMS products well-suited for the needs of the persistent-storage segment tend to use a different client–server model than ODBMSs designed for database management customers who need both production levels of multiuser performance and highly reliable databases. These customers tend to look for a balanced partitioning of DBMS responsibilities between client and server systems, e.g., with storage management, index management, buffering, physical logging, and query filtering at the server; and schema management, query optimization, transaction management, and object caching at the client. In these systems, the interface between the client and server components is at the object level. Another way to look at this balanced architecture is that the server part of the ODBMS has the responsibility to manifest objects from storage and present them to the client part of the ODBMS for delivery to the application. The trend is for the ODBMS server to be active in the execution of object methods as well.

By contrast, an ODBMS that is focused on persistent storage typically has a "skinnier" server and a "fatter" client. The server doesn't know much about objects; it manages page spaces and delivers pages to the clients who do the rest of the ODBMS-related work. This architecture imposes certain limitations that can adversely affect performance, e.g., query filters cannot be applied at the server. This can result in significant network traffic in multiuser systems because all candidate objects must be available at the client.

SOME ODBMS CUSTOMERS NEED NEW TRANSACTION AND DISTRIBUTION CAPABILITIES

A variety of ODBMS customers have applications that need groupwork functionality: versioning, long transactions, persistent locks, event notification, check-out and check-in, and so forth. These transaction-management and data- distribution capabilities simply are not available in conventional DBMSs or file systems. Examples of these applications include CASE for engineers who need to cooperate in the development of large applications, CAD to support concurrent engineering, process and financial simulation with extensive "what-if" scenarios, and document or design-management systems that simulate library environments.

> ***Evaluating the transaction management and distributed functionality of ODBMS products is not a trivial task.***

Customers building these applications are faced with either developing the groupwork functionality themselves (on either a conventional DBMS or a file system) or buying it via an ODBMS product. Increasingly, they are choosing the ODBMS approach.

Developers of these kinds of applications need to make sure that the ODBMS selected will enable their application to match the way their business operates, rather than tailoring business operations to the functionality of the technology. Some may need to be able to arrange hierarchies of databases, with controlled movement and sometimes replication of objects between the levels of the hierarchy. Some may want to be able to version objects with appropriate degrees of flexibility about which versions are usable in particular circumstances. And they commonly want to be able to configure heterogeneous network environments that scale with business needs.

EVALUATING GROUPWORK FUNCTIONALITY IS NONTRIVIAL

ODBMS products in both the persistent-storage and database management categories offer groupwork functionality. Evaluating the transaction management and distributed functionality of ODBMS products is not a trivial task. The challenge is complicated by the fact that vendors do not agree on the terminology used to describe features in this area. For example, you can't just look to see if a product supports versioning or not; you really have to dive down into the details of the versioning capabilities to get a fair comparison of the products. Similarly, the terms "check-out" and "check-in" refer to different sets of features for the various products. And to complicate matters further, typically there are subtle interactions among the groupwork features. For instance, check-out may or may not have versioning implications, depending on the product. Not all vendors offer persistent locking, which is essential for long-transaction recovery. Not all ODBMSs support nested short or long transactions. In general, ODBMSs focused on database management customer needs are more robust (i.e., suitable for production multiuser systems) in the area of groupwork functionality than ODBMSs for the persistent storage customer.

continued on page 18

tem, resulting in a typically awkward and code-consuming interface. ODBMS products offer an attractive alternative and move the customer from using a conventional persistent-storage solution to an object-based one.

Because of its tight integration with the object programming language, an ODBMS enables programmers to build their applications faster. In fact, for many programmers in this category, the more invisible the ODBMS the better. They may even view the combination of the object programming language and ODBMS as simply a "persistent-object programming language."

Not only is there significant reduction in the number of lines of interface code, but there also can be substantial performance improvements. One of our customers found its application ran 3–10 times faster (depending on the particular benchmark measurement) using an ODBMS instead of UNIX files.

CUSTOMER PRODUCT EVALUATIONS FOCUS ON SINGLE-USER PERFORMANCE

Once the decision to use an ODBMS has been made in the persistent-storage segment, the primary technical criterion in evaluations is usually single-user performance. Different customers use different performance benchmarks, of which few have been published. Most tend to emphasize in-memory access speeds, although some take a broader view to consider paging behavior. Customers hope an ODBMS essentially will deliver objects at programming language in-memory speeds. They generally don't care about concurrency and transaction-management features. By implication, an ODBMS aimed at this segment may de-emphasize these features in favor of single-user access speeds.

> ***The more complex the application is in terms of data types and interrelationships, the more likely that ODBMSs will deliver needed performances while relational solutions will not.***

It is important that customers having performance as their primary product-selection criterion use benchmarks that truly reflect their application's characteristics. This can be easy for relatively simple applications, where the application itself can be the benchmark, but it can be quite complicated in other cases, especially if over time the application changes the population of objects through extensive deletions and insertions.

OTHER CUSTOMERS NEED DATABASES OF OBJECTS

Increasingly, ODBMS sales are being driven by applications that require production levels of object sharing rather than as a consequence of adopting an object programming language and wanting persistent storage. These applications require database management functionality and have performance requirements that RDBMSs simply can't satisfy. Database management functionality is much more than object persistence.

Often these applications will execute in client–server environments, with multiple client machines accessing object databases on one or more server machines. Examples of these applications include geographic information systems (GIS) with extensive, non-tabular data; network management systems with complex interrelationships among many kinds of components; hospital information systems with many different kinds of data to profile patients and treatments; and CAD software with complex product-definition data.

FOR SOME, MULTIUSER PERFORMANCE IS NUMBER ONE

Some customers in the database management category have multiuser runtime performance as their primary reason for using ODBMSs. The complexity of their applications makes RDBMS performance unsatisfactory and moves these customers into the object part of the database management market.

Complexity comes in two flavors: (1) complex data types (like product designs, multimedia, images, voice, . . .) that don't fit well into relational tables and lose their internal semantics when stored as relational binary large objects (BLOBs), and (2) queries that require combining several kinds of data and result in low-performance multiway joins in a relational system. The more complex the application is in terms of data types and interrelationships, the more likely that ODBMSs will deliver needed performance while relational solutions will not.

Many of these customers are independent software vendors (ISVs) building the "next generation" of their products in the CAD, CASE, or GIS areas. These object-oriented products will replace single-user versions that are based on home-grown file structures. The buyers of these products want shareability, which the ISV can either build or acquire off-the-shelf with an ODBMS. Essentially, the ISV can decide to enter the database business or buy the functionality and performance from an ODBMS vendor. If an ODBMS is used, it must be able to deliver objects with runtime performance at least as fast as a home-grown file-oriented solution while also providing shareability.

FOR OTHERS, PRODUCTIVITY IS FIRST

Other customers in the database management market say that their primary requirement is to build the application faster, which I would restate as "productivity," although few customers actually use that term; they nearly always list performance as their second most important reason for using an ODBMS.

Applications in the object database management category are

There are also several nontechnical reasons why this "pull model" of reuse is the most likely to succeed commercially. Although the reward structure still needs improvement, the team that needs the class library is most likely to be rewarded for reusing classes (because of lower development costs, higher-quality result, etc). A team responsible for generating reusable libraries is rewarded for the amount of code generated, not for generating useful code. If it were possible to make money selling class libraries, competitive pressures would improve the situation significantly. Another problem with the push model is that classes are often developed in a vacuum, without a specific need in mind. The result is often overdesigned or unfocused code not useful for anything.

Keyword selectors Keyword selectors (available in Smalltalk and Objective-C) increase programmer productivity. If the developer must consult a manual (or, worse, the target code) to determine the meaning of the parameters each time an unfamiliar message is to be sent, productivity will be low. However, with keyword selectors and good naming conventions each parameter is labeled with a hint about its meaning.

For example, sending a message to a real number object to format itself onto a file stream in C++ might look like this:

```
realNumber.format(aStream, aFormat)
```

However, using keyword selectors in Objective-C, this becomes slightly more readable (though more verbose):

```
[realNumber formatOn: aStream using: aFormat]
```

The extra clues contained in keyword selectors are often just enough to allow an experienced developer to guess the meaning and order of the parameters without consulting the documentation.

Memory management Automatic memory management has received a great deal of attention lately. Many believe it is required for true object-oriented programming. I disagree. It is more likely that *manual* memory management methods will result in more reliable code, for two reasons. First, manual memory-management techniques impose the responsibility of "ownership" of all objects on the architecture so they can be freed at the correct time. The result is a more organized structure that has been necessarily partitioned very carefully according to responsibilities. Second, very subtle but potentially serious bugs can be masked by automatic memory management but are very obvious under manual memory management.

Consider a system without automatic memory management. Two objects, A and B, are designed to always share a third object C. If A is the owner of C and is instructed to throw away C and share object C1 instead, it will (correctly) free C. However, if there is a defect in the system that leaves object B with an invalid pointer to the old object C, the system will most likely crash with a fatal internal error. Proponents of automatic memory management insist that this problem is solved with memory management because the old object C will not be freed but rather will remain in existence as long as B has a pointer to it. However, I would say that in some cases this situation is even more dangerous than a crash because the system might silently report the wrong answer when an old, unused object (C) is being accessed by object B.

One important benefit of using automatic memory management is protection against memory "leaks" (allocated memory that is never freed). This potentially serious situation slowly consumes all available memory until the application runs out of memory and crashes. One way to solve this problem is to design the system with manual memory-management techniques but have a garbage-collection mechanism as a backup to protect against memory-leak defects. Another way to handle this situation is to spend some testing time studying memory usage to determine if memory leaks are a problem. Tools have been developed that monitor all memory allocations and deallocations to determine if memory is allocated but not freed.

C++ automatically copies objects in a variety of situations. This can be convenient in many situations but, because it does not support automatic memory management, memory leaks are a serious threat.

Operator overloading Operator overloading is a very good way to reduce the verbosity of message calls. It allows standard expression operators such as + and = to be implemented as message calls and therefore allows new data types to be defined that appear to work directly in expressions. This is not really an issue with "pure" object-oriented languages such as Smalltalk because there is really no base language. Every expression is already a series of messages and operator overloading is a natural way to make the language more usable. However, with a "hybrid" language such as C++ the base language is compiled directly into machine code but operators defined as messages are not. Making expressions of built-in and defined types look the same can often simplify the appearance of the source code but can also hide the message-passing inefficiencies.

Execution speed In the past, efficiency was a major concern when selecting a language. Now that incredibly fast machines are available at a low cost, however, this is no longer as important an issue. It is likely that the differences between various languages are insignificant when compared with the effects of a poor overall software architecture. Some real-time systems recently have been developed using very efficient implementations of Smalltalk. Still, there are some applications extremely sensitive to execution speed and the only choice is a high-performance hybrid language such as Objective-C, which has many of Smalltalk's benefits but its base language, C, can be used to attain the ultimate performance.

Objective-C allows the developer to improve performance of message passing by declaring statically bound messages. C++ does this by default; the normal object-oriented requirement of dynamic binding is the special case. Both languages allow direct access to instance variables. These speed hacks should never be used—they are only necessary to improve the performance of a poor architecture.

Runtime type determination Some languages allow access to the type information at runtime. Smalltalk uses this information

to support inspectors to help with locating defects. Objective-C also has this type information available at runtime, but only the NeXT platform utilizes this information for interactive tools. C++ does not have access to type information at runtime, so these debug tools are often difficult to implement.

Runtime type information can also be used for more advanced capabilities such as transforming an instance of one class into an instance of another. This is certainly not critical to a project's success but it can be useful.

Multiple inheritance Multiple inheritance is rarely the best way to implement commercial quality software. The standard examples used to support the need for multiple inheritance are trivial and do not reflect the real world. Multiple inheritance is useful for minimizing duplication of code. However, the goal is high-quality software, not the minimum number of lines of code, so simplfying the overall architecture is the most effective answer. Inheritance can simplify the architecture by accentuating the similarites between related classes, but it can also increase complexity due to additional dependencies on members of the inheritance hierarchy. Multiple inheritance magnifies inheritance's problems without really solving any of them.

In C++, true polymorphism is not supported but pseudo-polymorphism can be obtained by using an abstract base class to declare a general protocol. The problem with this technique is that often objects must take on the behavior of more than one abstract class. Multiple inheritance can be used to solve this problem, but a better answer is to select a language that supports true polymorphism without placing restrictions on the developer.

Standard class libraries The availability of standard class libraries and other runtime support is most important for very small applications and least important when developing large applications. The reason for this should be obvious: in a large development effort, the incremental cost to implement classes that might have been available in a library is small when compared with the total cost of the project.

Pre- and postconditions Some languages such as Eiffel support explicit preconditions and postconditions. Proponents claim that code correctness is guaranteed if these conditions are fully specified. That may be true but the problem is to fully specify all of the necessary conditions. Although these conditions do no harm, it is very unlikely that all necessary conditions can really be specified correctly and completely. In any case, defects are usually caused by incorrect specifications, not by incorrect implementation of the specifications. Therefore, in many cases the conditions will be specified incorrectly but the code will match the conditions exactly.

Development environment considerations In addition to features of the language, the development environment must be considered.

In some situations, very fast edit-test cycles are an important consideration. An interpreted language such as Smalltalk is the best way to reach this goal. However, experienced developers using a language that is not error-prone can be productive even if this cycle is not optimally fast because they generally write the code correctly the first time without needing to verify it.

Debug and verification features often can have a significant impact on the amount of time and effort spent during the testing phase. The availability of standard debug features, such as breakpoints on code or changes to data, etc., is important as are features specific to object-oriented development such as interactive inspectors, breakpoints in specific instances of classes, etc. With enough debug features, it is relatively easy to locate the source of defects in a large object-oriented system.

In a commercial development environment, there are several very important issues related to large-scale software development, such as source-code control, audit trails, and shared access to a central source-code database.

Sometimes there are commercial delivery issues such as licensing to overcome for the use of runtime libraries or interpreter, support for specific platforms, or memory requirements of the minimum system.

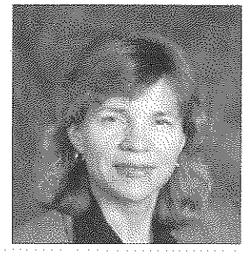
CONCLUSION

The selection of an object-oriented language involves comparing the goals and requirements of the project with the features and benefits of the various languages available. Smalltalk is probably the best choice based on language features and development tools but there are lingering issues for large development teams and the number of supported platforms is limited to those important to Smalltalk suppliers. Objective-C is a reasonable alternative to Smalltalk and is supported on many platforms. Because it is translated into C, code can be ported to any platform that supports the C language using cross-compilation techniques. Objective-C has many Smalltalk-like features such as typeless objects and keyword selector syntax for messages. Its hybrid nature also allows code to be written with the efficiency of C. C++ features such as virtual functions, automatic copying of objects, etc., place a heavy burden on the developer and, consequently, programmer productivity is limited. In addition, developers who do not fully commit themselves to the object-oriented approach may slip back into their old ways more easily with C++. This is usually not obvious when it occurs and can become a significant management issue.

Don't choose a language based on the amount of training required of your team. An object-oriented approach requires a new way of thinking about the problem and must be approached as such. Choosing a language just because it resembles one already being used spells disaster if the team is merely shoehorning a traditional approach into the object-oriented structure. ≡≡

Bill Hunt is a software-development engineer at Hewlett-Packard's Measurement Software Division in Loveland, CO. He develops software for test-and-measurement applications using object-oriented techniques. He can be reached via phone at 303.679.3642, via fax at 303.679.5957, or via email at billh@lvld.hp.com.

Object database technology: who's using it and why?



Mary E. Loomis, Ph.D.

ODBMS MARKET IS YOUNG AND GROWING RAPIDLY

Many observers of the object database management system (ODBMS) market consider it still to be in its infancy. The first ODBMS product was shipped by Servio Corporation in 1987 and most of today's "major" ODBMS vendors have been shipping product for only two years or so.* In 1991 the database management system (DBMS) market was approximately $4.12 billion. While ODBMS products represented only a small fraction of this total, it still amounted to a healthy market of about $10 million.†

Industry watchers expect the market for ODBMSs to grow dramatically. Projections generally are about $35 million in 1992 and nearly $100 million in 1993. The current annual growth rate for relational DBMSs (RDBMSs) is about 17% while the ODBMS market is growing at a 350% clip. James Martin projects that 50% of all DBMS sales in the second half of this decade will be for ODBMSs, not RDBMSs.

What a great time and place for technology entrepreneurs! New companies continue to enter the ODBMS market so the shake-out hasn't really begun. The most active ODBMS vendors, their products, and the programming languages they support are shown in Table 1.

WHY DO PEOPLE BUY?

People buy ODBMS products for a variety of reasons; no single-value statement applies to all ODBMS purchases. I want to share with you an approach I've found useful for categorizing ODBMS customer needs and I'd appreciate your feedback.

There are two basic dimensions to this categorization approach:

1. object vs. conventional
2. persistent storage vs. database management

A persistent storage mechanism is designed primarily to make it easier for users of a programming language to access disk storage for their data and structures locally or on a network. A database management system is designed primarily to provide multiuser data sharing to a variety of applications and programs locally or on a network. Persistent storage and database management needs to exist in both the object and conventional technology worlds.

ODBMS products are object based; some are more appropriate for persistent storage needs and others for database management needs. RDBMS products meet conventional database-management needs; file systems (such as those provided by DOS or UNIX) meet conventional persistent-storage needs. For now, let's focus on object-related needs.

SOME CUSTOMERS WANT PERSISTENT STORAGE OF OBJECTS

Customers in the object-based persistent-storage category buy an ODBMS because they are using an object programming language and need to create objects that will outlive the termination of program executions. Many of these sales are to individual programmers or projects. The applications are commonly single user and run solely within the domain of a workstation or PC.

Before deciding to use an ODBMS, object programmers may have implemented their own persistent storage using the file sys-

Table 1

Vendor	ODBMS Product	Programming Languages
BKS Software	POET	C++
Hewlett Packard	OpenODB	HP OSQL
Itasca	ITASCA	C, C++, Lisp
02 Technology	02	C, C++
Object Design	ObjectStore	C, C++
Objectivity	Objectivity/DB	C, C++
Ontos	ONTOS DB	C++, Object SQL
Raima	db_VISTA III	C, C++
Servio	GemStone	C, C++, Smalltalk
Versant Object Technology	VERSANT	C, C++, Smalltalk, Eiffel, Object SQL

* Compared with RDBMS vendors, ODBMS vendors are quite small.

† source: Forrester Research, Inc.

hotline on OBJECT-ORIENTED technology

Back issues

All back issues of the HOTLINE are available. Please call 212.274.0640 for details.

Vol. 3, No. 12/October '92 ≡ Development environments for the lifecycle ≡ Object methods: Selecting the right object-oriented method ≡ Languages: Choosing an object-oriented language ≡ Object databases: Object DB Technology: who's using it and why?

Vol. 3, No. 11/September '92 ≡ Developing strategic business systems using object technology ≡ Object training: harder than it looks ≡ Object-oriented ROI: extending the CRC across the lifecycle ≡ What TQM means for OT

Vol.3, No.10/August '92 ≡ Object technology: toward software manufacturing ≡ Return on investment: software assets and the CRC technique ≡ Object-oriented technology in Japan ≡ Providing commonality while supporting diversity

Vol.3, No.9/July '92 ≡ OOD: Research or ready ≡ Enterprise modeling: an object approach ≡ OMG's 18–24 month view ≡ Design for object-oriented applications: a CASE for wishful thinking. . .

Vol.3, No.8/June '92 ≡ Business in the Information Age ≡ From data modeling to object modeling ≡ How frameworks enable application portability ≡ Interview with Vaughan Merlyn

Vol.3, No.6/April '92 ≡ Thinking the unthinkable: reducing the risk of failure ≡ Mitigating madness with method: first establish what you value ≡ Championing object technology for career success in the 1990s ≡ Objects and actions in end-user documentation

Vol.3, No.5/March '92 ≡ TA large-scale users' assessment of object orientation ≡ Report on the Object-Oriented COBOL Task Group ≡ Interview with K.C. Branscomb

Vol.3, No.4/February '92 ≡ The big prize: acceptance of O-O by the MIS community ≡ Retrospective: 1991—the year it all changed ≡ Making the transition to O-O technology ≡ Interview with Beatriz Infante

Vol.3, No. 3/January '92 ≡ Enterprise object modeling: knowing what we know ≡ Adopting objects: pitfalls ≡ Adoption rate of object technology: a survey of NSW industry

Vol.3, No. 2/December '91 ≡ Accepting object Technology ≡ Adopting objects: a path ≡ Incorporating graphical content into multimedia presentations

Vol.3, No. 1/November '91 ≡ Leading the U.S. semiconductor manufacturing industry toward an object-oriented technology standard ≡ Coping with complexity: OOPS and the economists' critique of central planning ≡ Choosing Object Technology: What's the object? ≡ OOP: the MISsing link

Vol.2, No. 12/October '91 ≡ A modest survey of OOD approaches ≡ What is a "certified" object programmer? ≡ Perspective: investing in objects today ≡ Object oriented in Melbourne, Australia ≡ The Object Management Group

Vol.2, No. 11/September '91 ≡ From applications to frameworks ≡ Report on the Object-Oriented COBOL Task Group ≡ Getting started with object technology: efffectively planning for change ≡ Object statistics on the way ≡ On objects and bullets

Vol.2, No. 10/August '91 ≡ Distributed object management: improving worker productivity ≡ Getting the best from objects: the experience of HP ≡ APPLICATIONS: EC employs object technology ≡ CAPACITY PLANNING: Fiddling while ROMs burn

Vol.2, No. 9/July '91 ≡ Multimedia is everywhere! ≡ Developing an object technology prototype ≡ Object-oriented capacity planning ≡ How OOP has changed our developmental lifecycle ≡ Modularization of the computer system

Vol.2, No. 8/June '91 ≡ Domain of objects: the Object Request Broker ≡ Object-based approach to user documentation ≡ Report on the Object-Oriented COBOL Task Group ≡ Do we need object-oriented design metrics?

Vol.2, No.7/May '91 ≡ Hybrid object-oriented/functional decomposition for software engineering ≡ So, what makes object databases different? (Part 4) ≡ Using the generic application to solve similar domain problems ≡ Experiences using CLOS ≡ International Conference on Object-Oriented Technology, Singapore

Vol.2, No.6/Apr. '91 ≡ An artist's perspective of programming ≡ So, what makes object databases different? (Part 3) ≡ Moving from Pascal to C++, Part 3 ≡ Object projects: what can go wrong ≡ Reflections from LOOK-'91

SUBSCRIBE NOW TO THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY— DON'T MISS ANOTHER VALUE-PACKED ISSUE!

❑ *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)*

(outside US add $30 per year for air service)

Back issues @ $25 each *($27.50 foreign):*

Vol.2, Nos. _____

Vol.3, Nos. _____

❑ Phone/fax order
Call Subscriber Services at 212.274.0640
or fax this form to 212.274.0646

❑ Bill me

❑ Check enclosed
Make check payable to the HOTLINE and mail 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)

❑ Credit card orders
❑ MasterCard ❑ Visa ❑ AmEx

Card# Expiration Date

Signature

Name

Title

Company/Mail Stop

Street/Building#

City/Province

ST/Zip/Country

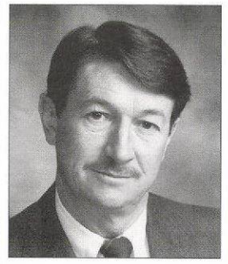
Telephone

D2KC

hotline on OBJECT-ORIENTED technology

VOL. 4, NO. 1 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES NOV. 1992

Combining object technology with data standards for the next industrial revolution



Thomas Rafferty

Interchangeable parts, assembly line production, efficient transportation infrastructure: without these critical developments, the first industrial revolution might never have taken place. Today, new developments in computer hardware and software are setting the stage for a second industrial revolution. Two emerging software technologies, when combined, hold great potential for accomplishing this transformation within the manufacturing industry: object-oriented systems and a standard, vendor-neutral format capable of supporting a product throughout its lifecycle.

As anyone working in the manufacturing industry today can tell you, improving "time-to-market"—the time it takes to design, develop, manufacture, and ship a product—is one of the most crucial issues for companies aspiring to compete in a global marketplace. As a result, manufacturers must continually search for tools that will provide them with a competitive edge; tools that will significantly reduce time-to-market without sacrificing product quality.

Imagine an environment in which manufacturing engineers have immediate access to design information; where parts designers can effortlessly call up the latest version of related components; and where support personnel can quickly retrieve, test, and repair records for each product, part, and component. This networked environment would enable separate design teams, whether internal or among subcontractors, to work cooperatively and in real time regardless of geographic location.

Computers offer a great potential to reduce time-to-market by providing the means for such collaborative efforts. Until now, however, companies have been hampered by proprietary software systems that make information sharing with outside vendors both costly and difficult. Even within their own firms, existing software systems have not adequately addressed the growing need to share accurate and timely information among different divisions, departments, and disciplines.

Today the design, manufacturing, and business management functions of an organization must work in tandem and have the most effective tools if the company is to remain competitive. For computer-based systems, this means software that provides a flexible infrastructure that can seamlessly integrate a variety of applications, both existing and new, into a global network of engineering and business information systems.

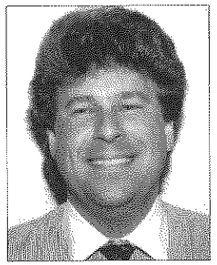
HOW TO GET THERE FROM HERE

Phenomenal price/performance strides have been made by hardware vendors during the past few years. The potential for equally

continued on page 4

IN THIS ISSUE

FROM THE EDITOR



Today, I enjoyed yet another insightful conversations with a friend who has worked with a leading clothing retailer for over a decade. He has seen the firm grow from 100 million to one billion in sales. Along with his teammates, he has struggled to maintain the large mainframe business applications that keep products rolling out the door to satisfied customers day after day, year after year. He has seen technologies come and go—and been one of the people whose job it was to make these technologies live up to expectations. Today, my friend asked me about objects.

Why is this new technology sounding more and more like the previous, only with a new vocabulary? he asked. How can I expect management to be interested in a new set of buzz words? Why do the vendors all push implementation-level concepts when explaining object technology? Don't they understand that implementation is not where the problem is? What does this programming language concept really mean at the business enterprise level? How is any of this different from what information engineering was supposed to be?

Would that I could provide good answers in so few words! No fewer than a dozen IT managers from billion-dollar-plus firms have asked me similar questions in the last few weeks.

Well, to start with, my friend's basic observation is right. Object technology *has* grown out of programming language constructs. What is important about *some* of those constructs is that they keep showing up in other disciplines, and in the way we observe the world around us. **Mr. Thomas Rafferty,** Vice President of Auto-trol Technologies, makes the value of this parallel clear in his feature article this month about objects and manufacturing. Objects make intuitive sense because we can make analogies between objects and the real world. To business managers this means They can expect us to build systems with objects that look one hell of a lot more like their world than ours. Better, but only a start. Right though we are, we are skirting the real question.

Over the last twenty years, our industry has evolved approaches to developing computer-like software for business people. Computer technologists and business people alike have long viewed this state of affairs as immutable reality. Only in the last few years have business people experienced the insight of working with analysts who model the real business of the business. Some of these business people have made a clear distinction between the value of modeling computer systems (a technical discipline used in blueprinting for software development) and the value of modeling the business (a business discipline that often involves IT professionals). These people gain business value from such technologies as information engineering, business process reengineering, and objects. But, often through no fault of their own, these people are the exception.

Although Mr. Edward Yourdon in THE DECLINE AND FALL OF THE AMERICAN PROGRAMMER observes that the state in which IT finds itself today with CASE and methods "...does not mean that there is something wrong with structured methods...", I suggest that my friend's experiences on the job—and yours and mine as well—tell us that the problem started right there! From the get-go, many of us were taught to conform the business to computer-based models. We were taught that business processes should decompose into two to six subprocesses. We were taught that the business processes at some point (called design) suddenly began to decompose further into completely custom-built computer logic. Last, and most costly, we learned that applications controlled the data—so today businesses are spending billions of dollars trying to reassemble into shared structures the very business data that we spent several decades decomposing into separate and disjoint islands. Even Dr. James Martin's information engineering failed to free itself completely from the bondage of the computer system. How strong is the grip of iron!

Don't misunderstand me. Without the discipline of formal methods, high-quality systems cannot be built and maintained cost-effectively. Quality does not just happen, it must be built in from the start. Mr. Sam Adams reinforces this point eloquently in this issue, in the closing article of his insightful four-part series. I think we all agree that engineering discipline is critically missing from too many IT investments. This, however, does not mitigate the business impact of our heritage. Of the developers who have been educated in structured or IE approaches, all too many seem to not understand the difference between modeling the concepts (problem space) and the implementation (solution space). Machine-based models and constraints are still driving our business applications. Our software has been constraining the business, its growth and flexibility. To succeed in the IT market, the object technology industry will have to show how objects can be used to change this situation.

Can we focus our efforts with this technology on the areas that our industry has traditionally missed? What object-oriented method is directed at something other than applications design or analysis? How many object-oriented methods provide substance in *both* analysis and design? How many effectively handle the transition *from* analysis *to* design? How many contenders help the IT professional glean objects from their inventory of business models? How many address sharing information, distribution of data, distribution of processing, operations management, application performance assessment, quality control, security and anti-viral imuno-system strategies, business policy and semantics, global network management, peer-to-peer applications where client and server are relative concepts, design for zero-cost portability, design from components, and TQM? Ok, is my point clear?

Ms. Patti Dock of OrgWare continues to examine many of these questions in her column and finds, for example, that the Jacobson method is one of the few that addresses testing and software components. Testing and quality are an especially sticky wicket, as you will see in reading Mr. Adams discussion of the subject. Both business and IT stand to reap substantial rewards from the wide-scale reuse of well-

of OOP. But more important, at no time did this person grasp that reusability—one of the great benefits of OOP—might be possible without cutting and pasting code. Although I'm a big fan of FoxPro (and Fox Software), I was surprised at how many additional misconceptions and misunderstandings about object-oriented programming were being perpetrated. . . .

C++ and client/server, Richard Hale Shaw, **DBMS**, 7/92

LANGUAGES

. . . In most languages, learning to program means learning the syntax. Learning to program in Smalltalk, however, involves much more. The programmer must have a clear grasp of object oriented concepts. In addition, Smalltalk's development environment strongly influences the entire approach to software creation. It is absolutely essential that the developer become familiar with the classes provided by the Smalltalk environment. Although this can take some effort, it's a prerequisite for developing more than the most trivial programs. Fortunately, this is an interesting activity and is one of the best ways to learn Smalltalk. . . .

. . . The Smalltalk development process usually involves rapid prototyping. The typical approach is to quickly implement a rough version of part of the system so you can begin to get feedback that will shape later development. For this reason, the user interface is usually the first part of a Smalltalk-based system to see the light of day. Additional methods are fleshed out as needed. The more difficult parts of the program are often left until the end, when the rest of the supporting structures are in place. Numerous developers use this approach to quickly deliver software. . . .

An earful of Smalltalk, John D. Williams, **PCAI**, 9-10/92

. . . OOP is not a panacea. Small's second law of software, "You can write a bad program in any language," applies equally well to C++. A fanatical OOP programmer can compress and condense his source code wonderfully. But at compile time the compact source code could expand into a cumbersome plodder of a program. OOP deliberately emphasizes the relationships between things while suppressing the processing that operate on these things. Thus, you could lose sight of how things work. Learning the new syntax of C++ is not difficult if you are already familiar with C. . . But changing your mindset from procedural to object-oriented programming takes time; six months is a common estimate. Breaking a problem down into a hierarchy of objects is far from an exact science. And no method exists to test how optimal a given breakdown is. . . .

. . . Proponents fervently hope that C++ will lead to more code sharing and reuse. While a given programmer may be able to reuse his own code more easily, code sharing within and among companies depends on more than a good language. A host of organizational, territorial, legal, financial, and archival problems overshadow the problems of grafting a foreign bit of code into a program. While Ref2 is ostensibly about reusing Ada code, the article actually describes the tough, realworld problems of code reuse that have little, if anything, to do with computer languages. . . .

How C++ works, Charles H. Small, **EDN**, 8/6/92

Apple Computer Inc. is working on a new language that it hopes will bring simplicity to application development, much as the Macintosh brought ease of use to desktop computing. The new language, which Apple classifies as an object-oriented dynamic language, or OODL, is called Dylan. With roots in C++, Smalltalk and Common LISP, Dylan is Apple's attempt to meld the features of static and dynamic languages into a new paradigm of programming. . . Apple plans to license Dylan to third-party tool vendors and will build its own application framework around the language. . . [and]also plans to use it to develop applications for its Newton personal digital assistant. . . in order to compete with C and C++, Dylan will have to prove itself as a portable language, one analyst said. "It's an absolutely easy language to write to—the problem is portability," said Jeffrey Tarter, editor and publisher of Soft•letter, an industry newsletter in Watertown, Mass. "[Apple's] talking about essentially a proprietary language, and that's going to be a hard sell in a world that's moving very rapidly toward C and C++ and other highly portable products."

Apple's new language: it's OOP from the ground up, Cara A. Cunningham, **PC WEEK**, 8/17/92

IBM has officially changed the plan for AD/Cycle as an acknowledgement of the expanding role of workgroup computing in software development. The new plan calls for incorporating the Repository Information Model on AIX and OS/2 via a third party-supplied object-oriented database, said Jon Hemming, IBM manager for market strategy in the programming Systems unit in Somers, NY. The OODBMS has not yet been selected. . . .

Workgroup advances change ad/cycle plan, SOFTWARE MAGAZINE, 8/92

STANDARDS

. . . In developing Cairo, Microsoft's object-oriented version of Windows due in 1994, the company thus far has also bypassed standards work being done by the Object Management Group (OMG), an industry consortium that includes Microsoft and nearly every other systems-software provider. . . Microsoft has not worked with the OMG to ensure that the new object module in Cairo will comply with the object-model standards being developed by the OMG. . . .

Microsoft bets its future on NT, Paul M. Shere, PC WEEK, 8/17/92

. . . The American National Standards Institute (ANSI) has gained the support of the vendor-sponsored Object Management Group (OMG). . . in its effort to head off any divergence in standards for object-oriented technology. The X3 Committee, authorized by ANSI and managed by the Computer and Business Equipment Manufacturers Association (CBEMA), Washington, DC, established a new committee to bring together object-oriented precepts for programming languages, database services, user interfaces, open systems interconnection profiles, open distributed processing systems, operating systems interfaces and information modeling standards. . . William Kent, a software engineer in the Hewlett-Packard Co. laboratories in Palo Alto, Calif., and chair of the new committee, said the X3 group was forced to act quickly as software components and standards are being built on fundamentally different premises and object-oriented concepts. Each is justifiably called object-oriented, he said, but the different software packages still cannot communicate with each other. Kent contends that the many definitions are currently diverging rather than converging. . . .

OMG backs ANSI effort, Henry Heffernan, SOFTWARE MAGAZINE, 8/92

STRATEGIES

In the roughly 23 years since the development of Simula 67, the grandmother of object-oriented programming languages, object-oriented software concepts have achieved wide endorsement. They are getting applied to operating systems, libraries, programming languages and particularly user interfaces. But the focus on user interfaces—a highly visible but not very profound application—may mean that object-oriented concepts' deeper, more valuable a potential for addressing the problems of open systems could go undeveloped. . . .

Objects: Not just another pretty interface, L. Peter Deutsch, ELECTRONIC ENGINEERING TIMES, 8/24/92

. . . Design, coding, debugging and management of complex class hierarchies of C++ did bring some additional challenges to the programmer. However, the rapidly improving programming tools, substantial sources of class libraries and evolving operating systems have allowed class libraries to intermingle with standard APIs and foundation classes, mitigating much of the challenge of object-oriented programming.

As a minor participant in the creation of the Cobol legacy, I can empathize with the large intellectual investment in code and programmer training. But corporations that ignore the enterprise computing model and the new object-oriented architectures will be trapped between two different worlds. . . .

Entering the enterprise era, Ann Winblad, COMPUTER WORLD, 8/3/92

O-O DEFINITIONS

. . . At last fall's Fox Software Conference in Toledo, for example, I listened to a panel of database luminaries and Fox officials discuss OOP. One Fox spokesman, convinced that FoxPro was object-oriented, spoke of its inherent ability to let you reuse code: "All you have to do is generate well-designed, modular code and then cut and paste it from one module to another." In his view, this made such code "reusable" (albeit, in an extremely limited fashion) and therefore "object-oriented." At no time did this person mention inheritance, encapsulation, or polymorphism, which are considered the three primary tenets

tested components. The risk with Zultner's translation of Deming into software testing is that it sounds much like the concept of mass inspection that Deming so opposes. The distinction is a fine line, and to my knowledge neither Zultner nor any other practitioner has clearly defined how we will do statistical process control for software.

What our industry seems to miss repeatedly is that IT managers *live* with the above list of problems. These are not just future issues! These are issues that need to be addressed today—in our methods, tools, and approach to management. Application analysis and design is simple by comparison! Our industry is once again addressing only the relatively narrow technical scope that is easy to control, and not the broader-scope issues of how the applications we design must integrate into the businesses they are expected to support. It is easier to focus on applications than on the business and technical environment in which they are deployed. But how, in light of this, can we expect that the object technology industry will now focus on problems that were never found palatable before?

What I beseech my colleagues for is original thinking that addresses problems that could make a difference in the way we run businesses—right NOW! At present, we are addressing technical concerns first—once again—while simultaneously telling our business partners that objects will bring software closer to the business. Our actions and our words need to be brought into alignment. Sure, objects do allow us to represent business concepts more effectively in software, but vendors have to show IT how to deliver that! This is what Mr. Adams addresses with constant quality management, and what the AMiX component marketplace, described in this issue by **Mr. Howard Baetjer** and **Mr. William Tulloh**, must deliver. At the same time, we have to bring business closer to software development—taking the befuddlement out of the process of getting flexible, correct solutions to business problems; getting business input into the design of truly flexible systems built around shared business information; showing IT how to make large-scale reuse work at the level of business components, not just code; and showing IT how they can help their business partners make money from this powerful new wave of "open" technologies.

As Mr. John A. Zachman has often observed, working bottom up from programming and design constructs got us into the mess we face today with structured methods, IE, and CASE. We have to think beyond programming language constructs to answer the questions of business and IT professionals like my friend. **Mr. Timothy Andrews** of ONTOS addresses vendor involvement and commitment to this end in his monthly column about distributed information by first addressing ROI, not technology. The other side of reality's double-edged sword is that our business partners now question how much better off *they* are for funding us through these very expensive technology advances. When measuring their return on investment: business flexibility, data sharability, force reductions (and all the other benefits they expected from IT), they are not better off in *their* own eyes. That's the problem. That's the problem we should be addressing with object technology.

With object technology, we are raising good issues. As an insider, it is clear to me that, properly used and managed, this technology can bring substantial technical and business benefits. With the tools and concepts this industry is pushing, we can go where no developer dared go before. But the resounding questions coming from our business and IT partners are: when will we show them how and why it is different? When will we show them how to get there from here? Our industry's performance in showing IT how to succeed with object technology will make or break how IT perceives the viability of object technology over the next twenty-four months.

When so many IT managers are asking basic questions like those of my long-time friend, and our industry is so ill equipped to answer them, a red flag has been run up the pole. It's not "one if by land, two if by sea." It's both, and I frankly believe that most vendors severely overestimate the limited time they have available to understand and answer these questions with substance that will speak to the reality-hardened IT manager.

hotline on OBJECT-ORIENTED technology

Robert Shelton, Editor

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012, (212)274-0640. Mailed First Class. Subscription rate — one year (12 issues) $249, Foreign and Canada $279. Single copy $25.

POSTMASTER: Send address changes & subscription orders to HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville, NJ 07834.

Submit editorial correspondence to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of HOTLINE ON OBJECT-ORIENTED TECHNOLOGY, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, THE X JOURNAL, C++ REPORT, THE SMALLTALK REPORT, and THE INTERNATIONAL OOP DIRECTORY.

great strides in the software arena has exceeded the software vendor's ability to keep current. In an effort to retain the value of hundreds of man-years of software development, many CAD/CAM software developers have elected to build new capabilities (surface and solids modeling systems) on their old technology (wireframe design). Based upon traditional third-generation programming languages, these packages have grown into enormous, monolithic pieces of "spaghetti code" difficult to maintain and enhance. Many newer entrants into the CAD/CAM marketplace have elected to start "from scratch" and offer specialized software addressing the needs of niche markets. These new vendors have the luxury of inventing new automation processes and refining existing ones, but often do not have the expertise, funding, or time to develop the complete solutions needed by manufacturing companies fighting on the "front lines" of world competition.

The missing link in the evolution of total systems solutions is a software environment specifically designed to support the integration of applications, while providing a firm foundation for the development of a new generation of modular engineering applications. Such an environment would offer both traditional and niche software vendors the opportunity to tie their products into a network of applications working as a single unit to address specific customer requirements. The benefit of such an environment to customers speaks for itself: access to the software solution they were promised years ago; and the ability to build their own "shopping list" of applications without having to absorb the responsibility of making disparate products work as a single entity.

Object-oriented technology can provide such an environment. The most powerful example today is the emerging field of object management, the combination of distributed computing and object orientation. Distributed computing is an inherently flexible method of building systems by linking resources—computers, printers, applications, etc.—across networks. Object-oriented technology offers a framework for developing highly complex applications by constructing them from self-contained building blocks called objects that combine both data and functionality. As manufacturing processes become more distributed, the automated infrastructure must follow suit. Distributed object management is the means to this end.

DISTRIBUTED OBJECT MANAGEMENT

Distributed object management was originally conceived as a means of integrating diverse software applications, computers, and peripherals. With this technology, each component of an information system is defined as an object with clearly specified capabilities. Each object can then communicate with other objects to request actions, data, or services without knowing the other objects' locations on the network or their internal operations. Distributed object management systems provide the ability to encapsulate existing applications—everything from spreadsheets to advanced engineering applications and databases—by pro-

continued on page 11

STEP Fact Sheet

STEP is an acronym for Standard for the Exchange of Product model data, a standard designed to provide a mechanism capable of describing product data throughout the lifecycle of a product—through design, manufacture, utilization, maintenance, and disposal. Because this process involves data exchange between different computer systems, product information must be in computer-interpretable form. In addition, product data needs to remain complete and consistent throughout its use. STEP makes product data suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving across multiple computer platforms.

The goal of STEP is to allow and support concurrent engineering among design engineers, manufacturing, and support personnel by providing a standard mechanism for exchange of data between incompatible computer systems and applications. The development of this standard is critical to manufacturing companies which invested heavily in automation in the 1980s and today are struggling with data incompatibility to the detriment of teamwork among engineering disciplines, quality product delivery, and time to market.

The STEP standard is organized under the International Organization for Standardization (ISO 10303 Product Data Representation and Exchange). This is a voluntary activity among more than 15 countries. In the US, STEP is coordinated by National Institute of Standards and Technology (NIST), the successor organization to the National Bureau of Standards, under the IGES/PDES Organization administration at NIST.

Key aspects of STEP:

1. STEP defines the schema for data to be captured and exchanged including a definition of the data to be exchanged. In addition, tests can be generated to verify conformance to STEP.
2. STEP application protocols define the high-level items to be exchanged. Protocols defined include explicit drafting, associative drafting, configuration control and design, B-Rep solids modeling, surface modeling, wireframe modeling, process modeling and NC programming.
3. STEP captures all product data, including: product information, tolerance specification, material specifications, surface finish information, feature definition, shape, and others.
4. STEP captures product support information relevant to the product over its lifetime including analysis and test results, manufacturing process plans, setup sheets, tooling and NC information, QA inspection information, and support information.

For further STEP information contact:

William Conroy, The IGES/PDES Organization Office, NIST Bldg. 220, Rm. 127, Gaithersburg, MD 20899; 301.975.3981.
Haidee Rapacki, PDES Project Deputy Chair, Autro-trol Technology, Inc., 12500 N. Washington St., Denver, CO 241-2400; 303.252.2886.

FYI

Excerpts from leading industry publications on aspects of object technology

DEVELOPMENT & DESIGN

. . . The tasks in an object-oriented effort are different. New tasks are required to identify, characterize and document objects. These tasks focus on identifying objects and the interactions required of these object to provide a system that meets stated requirements. Object-oriented efforts, like other development approaches, need requirements and design specifications. Yet these documents localize around objects, and not functions or data. In addition, these specifications clearly delineate which components are reused from an in-house reusability library and which are developed from scratch to support the application at hand. Tasks associated with the construction of structure charts, data flow diagrams and other function- or data-oriented modules are obsolete and replaced with modeling approaches more in concert with object-oriented development . . . .

Designing the object-oriented way, Ron Schultz, OPEN SYSTEMS TODAY, 7/20/92

. . . No fundamental change in the pace of software development can occur until there is a significantly higher level of application development. In other words, end users must become developers. Object-oriented programming could allow end users to do just that. The ideal application development environment would consist of enormous libraries of prefabricated, modular program parts (super high-level objects). These modules could be configured and combined in virtually unlimited combinations to build complete applications across the entire spectrum of software use. Applications would be built exclusively in a high-level tool of this sort. Conventional code-level programming would focus on creating object components . . . End users would have unprecedented programming opportunities . . . .

The new shangri-la?, Joseph Firmage, SOFTWARE MAGAZINE, 7/92

DATABASES

. . . What will be the impact of OOP on database application languages and client/server technology? First, I expect C++ to be used as a model for object-oriented extensions to popular database programming languages. Indeed, early descriptions of the next generation of popular database products indicate that this is already in the works. Consequently, many of the benefits of C++—the capability to create new data types, inheritance, simpler, reusable code—should become *de rigueur* as new versions of database management systems appear. Future database management systems will be based on objects, and you'll be able to create generic, all-purpose components that you can reuse in your own applications, via inheritance . . . Applications will need very little "hard-wiring"—only to the type of objects used, not to a particular access method. "Soft" components will become the order of the day . . . You can also expect to see upcoming database management systems offering transparent access to different types of servers, via plug-in components—objects. These objects will provide polymorphic multiserver access methods that let a client application use a single approach for accessing disparate servers with differing protocols and commands . . . Finally, all the objects used in such a system would have persistence: the capability to save themselves to disk and restore themselves on demand . . . .

C++ and client/server, Richard Hale Shaw, **DBMS**, 7/92

THE BUSINESS OF OO

. . . Was it object-orientation (OO) that attracted Computer Associates to Nantucket Corp.? I think so. If you couple Clipper's use of OO with a healthy base of installed PC applications, you have a very attractive package . . . Clipper will evolve into more than just a DOS-based system, becoming operable under whatever operating system CA's other products run. Experts peg 90 percent of CA's sales as mainframe products. With PCs closing the gap to mini-mid-range systems, it isn't hard to see why CA may be in a better position than Borland Int'l or Microsoft. Although Borland and Microsoft are principally known in the PC marketplace, coming from the mainframe world, CA has a gigantic user base that wants to use PCs to their best advantage.

Why I think CA bought Clipper, Clesson M. Duke, DATA BASED ADVISOR, 7/92

Symantec Corporation Symantec Corporation announced **MultiScope Debuggers Version 2.0**, supporting Borland C++ and Microsoft C 6.0 and C/C++ 7.0 languages for programming Windows and DOS applications. Symantec entered into an agreement for the acquisition of MultiScope in June 1992. The DOS debuggers can be Windows-hosted and allow debugging of any size DOS application in a DOS window and offer full Windows 3.1 support. MultiScope Debuggers for Windows have a suggested retail price of $379.

Symantec, 10201 Torre Avenue, Cupertino, CA, 800.999.8846, 408.253.9600, fax: 408.253.4092

TauMetric Corporation TauMetric Corporation has an upgraded version of its **Oregon C++ Development System for SPARC**, offering the professional programmer direct code generation, C++ 2.1 compatibility, fast compilation, high-quality code, and a C++ debugger. The compiler generates object code directly; there is no translated output. Oregon C++ is currently available for VAX/VMS, SPARC, MIPS (DECstation), HP 9000/300, and Sun-3 Systems.

TauMetric Corporation, 8785 Fletcher Pkwy, Ste. 301, La Mesa, CA 91942, 619.687.7507, fax: 619.697.1140

Partnerships & Acquisitions

Object Design Inc. announced the formation of **Object Design Australia Pty. Ltd.**, a wholly-owned subsidiary based near Sydney, Australia. The new company, headed by managing director Philip Considine, provides marketing, sales, and technical support to Object Design's commercial accounts in Australia and New Zealand.

MetaWare announced that Bennett C. Watson, formerly Vice President of Technology at Ryan McFarland Corp., is taking over duties of current president Dr. Franklin L. DeRemer. Dr. DeRemer, who will continue as CEO and Chairman of the Board, assumes the role of Vice President of Business Development, working with MetaWare's cofounder Dr. Thomas J. Pennello.

HyperDesk Corporation and **Object Design Inc.** (ODI) announced the availability of an interface between their products. The companies have incorporated ObjectStore, ODI's object-oriented database management system, into HyperDesk's HD-DOMS distributed object management system.

Software Maintenance and Development Systems Inc. (SMDS) announced a cooperative marketing and product integration agreement with **CenterLine Software**, Inc. Under the agreement's terms, CenterLine and SMDS will participate in joint marketing and product integration efforts. This is one of a series of relationships CenterLine has developed under CenterStage, its third-party marketing program.

Informix Software Inc. licensed **Hewlett-Packard's** object-oriented database, Open ODB, to be integrated with its INFORMIX On-Line relational database. HP will also share with Informix other object-oriented technologies for distributed object-oriented environments to be integrated into the INFORMIX-OpenCase/ToolBus environment, based on HP's SoftBench software-development framework.

Data General Corporation and **NeXT Computer Inc.** will announce an agreement this month in which Data General will resell NeXT workstations along with its own Aviion servers. The agreement is designed to give Data General customers access to NeXTs object-oriented software.

The Object Management Group (OMG) announced that Petrotechnical Open Software Corporation (POSC) will include in its interface (API) specifications Common Object Request Broker Architecture (CORBA)-conformant technology. POSC has also endorsed the CORBA specification and ORB implementations as vehicles for providing a standard, interoperable framework to support these technologies in a heterogeneous environment. This agreement marks the first time an end-user organization officially endorses CORBA specifications. POSC is a not-for-profit membership corporation dedicated to facilitating industry development of integrated technology.

Constant quality management

(part 4 of a series)

Sam Adams

How do we maximize the benefits of object technology and manage the risks?" is a question raised throughout the IS community. Computing-dependent organizations face the challenges of downsizing, distributed computing, cooperative processing, GUIs, and the constantly changing application requirements of their users. This article is the fourth in a series concerned with meeting the challenges of enterprise-wide computing using object technology.

In previous articles we have defined the requirements for maximizing return on investment in object technology by creating and managing reusable software assets. We have overviewed KSC's approach to a lifecycle methodology for the development of object-oriented business systems from requirements to code and back. Tool requirements for the deployment of object technology on an enterprise scale were also brought into sharp focus. We now introduce *constant quality management*, a strategy and process for the development, reuse, and management of software assets. Using this approach, supported by fully integrated testing and constant metric feedback at all levels, the entire software lifecycle can be managed with confidence for consistent high-quality results.

ACHIEVING SOFTWARE QUALITY

High-quality software is designed to meet or exceed the needs of the user without violating user expectations.[1] In our experience, adhering to the following principles is critical to high-quality results:

- Maximize user involvement throughout the process. The user is the best person to determine if needs are being addressed and expectations met.
- Expect and encourage iteration throughout the software lifecycle. Only continual design validation, measurement, and refinement throughout the process can ensure that constant quality management is achieved.

In addition, the right decisions must be made at the right time. The earlier in the lifecycle a design decision is made, the greater is its potential impact, both positive and negative, on system quality. Lower-level design decisions made too soon can prevent the discovery of better solutions by prematurely limiting the number of alternatives, while deferred high-level "business" decisions may be too costly to implement.

In an industry where reuse must become the rule rather than the exception, methodologies and tools that don't explicitly support these principles are simply insufficient for the task. All aspects of the lifecycle, including people-oriented processes, tools, and methodology, must support constant management of produced deliverables' quality and value, including all analysis and design information as well as program code.

WHO'S RESPONSIBLE FOR QUALITY?

The software testing group? The project manager? The team leader? The users? In fact, everyone involved in the development lifecycle has some effect on the quality, whether positive or negative, of the delivered result. It is not a new idea that quality must be the daily responsibility of every participant in the process. Dr. W. Edwards Deming, an American statistician who went to Japan after World War II, taught this to Japanese managers over 30 years ago. He convinced the Japanese that they could deliver highest-quality products faster and at a lower cost than their competition. Since then, the maxim for Japanese industry has been "Quality that is taken for granted,"[2] with the assumption that customers have the right to expect quality products. Producing "Quality that fascinates" is the Japanese goal.[3] Customers should not just be satisfied with product quality, they should be delighted. A product should excite by exhibiting a level of excellence that demonstrates the company's passion for customer satisfaction.

The key to Deming's approach was not statistics, engineering discipline, or technology, but a devotion to continuous quality improvement.

THE DEMING WAY AND SOFTWARE DEVELOPMENT

Why not apply these same principles to the software development lifecycle? "We don't have the time or the budget for that level of quality" is often the reply. In the beginning, many Japanese managers felt the same way. But in the words of an old motor oil advertisement, "You can pay me now [for the oil], or pay me later [for the repairs]." Most IS organizations today spend the vast majority of their budgets maintaining systems that were developed years ago to meet minimum requirements. In reality, the "total cost of ownership" for those systems is many times the original cost of development. "Can we afford software quality?" The question is rather "can we afford the lack of it later?"

Richard Zultner, one of Deming's disciples, has applied The Deming Way to the traditional software development lifecycle.[4]

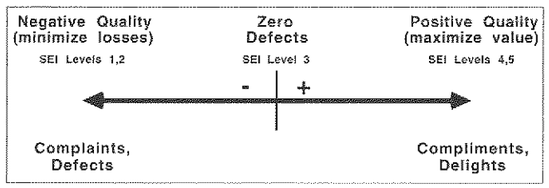


Figure 1. The quality continuum.5*

One of his specific recommendations for IS managers is to cease dependence on mass inspection, especially testing:

> The traditional thrust of software quality has been to use brute-force testing. Yet testing neither improves nor guarantees quality. Testing simply (and imperfectly) sorts code into two piles: "OK so far," and "rework." Rework adds delay and cost, but no value. Many project plans (and even methodologies) call for a massive effort to (try to) find and fix the large number of defects expected—even taking up an entire phase of the project to do so. What does this say to the developers about how many errors are expected? Is their job to build correct software—or to meet the schedule?[4]

How different is this perspective from that of continuous quality improvement! High-quality software simply cannot be developed using the "write the code, add testing, and stir" recipe. Most software development projects today are organized and managed around the goal of limiting (if not eliminating) the number of software defects delivered in the final product. While a better process than uncontrolled hacking, this focus on "Zero Defect" management can never lead to high-quality results (Figure 1). It can only produce minimally acceptable software, not software that excites the user. If organizations are ever to reach a state of continuous improvement, we must accept zero defect management as only a milestone on the path, not a destination.

Our focus should be on improving quality, not achieving zero defects. We must adopt a philosophy for software development that requires constant management of software quality and continuous improvement on the development process.

CLOSED LOOP CONTROL AND CYCLE TIME

Our approach for reaching these goals is based on a simple idea. The best time to assess the quality of any design decision is when you make the decision. In a process as complex as software development, the notion of "closed loop control" is critical to successful management and high-quality results. To illustrate this point, think about how you drive your car. Could you successfully reach your destination if you drove with your eyes closed? Would you even leave the parking lot? Visual feedback is a requirement for driving. But how much is enough? If we drove our cars like we develop software, we would think long and hard about where we wanted to go, close our eyes, and hit the gas. Then later, at a predetermined time, we would open our eyes and make sure we had arrived at our destination. Obviously no one would attempt to drive in this manner. What if you only closed your eyes for one minute out of every ten? No way. One second out of every ten? Maybe, but not with me in the passenger seat. The same problem would exist if you drove with your eyes open all the time but could only move the steering wheel every other minute. The issue here is not the time or effort spent seeing or steering, but the cycle time between each observation and course correction.

Philip Thomas is an expert on cycle time. He has helped hundreds of organizations drastically reduce the time it takes to produce products while simultaneously improving quality. He gave the following advice to the cellular phone division of Motorola:

> . . . suppose you built a feedback loop into each cycle: After you make 100 phones, you test them and find only 93 percent of them are good. You take the 7 percent and figure out, "If I'd done this differently, they'd have made it." Then you adjust your process. With a feedback loop built into each cycle, you improve something every time. That means: The shorter the cycle time, the more frequently you improve your product or service and the faster quality improves.[6]

> ***If we drove our cars like we develop software, we would think long and hard about where we wanted to go, close our eyes, and hit the gas. Then later, ...open our eyes and make sure we had arrived at our destination.***

Although software development is different in many ways from mass production hardware manufacturing, the same process of iteration can occur. The difference is that the iteration occurs on a single deliverable product, instead of the next batch of identical cellular phones. Thomas goes on to describe the results of closed loop control and reduced cycle time at Motorola:

> When Motorola was running a cycle every two weeks, that gave them only 24 opportunities to learn every year. Now they have 1,500 opportunities a year. They recently brought out the smallest, lightest cellular phone in the world—ounces less than the Japanese.[6]

CONSTANT QUALITY MANAGEMENT

We must apply these principles to maximize software quality throughout the entire development lifecycle. The best time to assess the quality of any addition or change to the requirements,

* SEI levels referenced in Figure 1 refer to the stages of corporate evolution toward the continuously improving organization. See Paulk[10] for more information.

Micro Data Base Systems Micro Data Base Systems, Inc. (mdbs) is now shipping the **Object/1 Professional Pack** for the Oracle relational database management system. Developed jointly by mdbs and Database Engineering Ltd., the Object/1 Professional Pack for Oracle allows developers to manage an Oracle session within an Object/1 application. Object/1 is an object-oriented development environment that allows rapid application development of graphical user interfaces (GUIs) in Windows and Presentation Manager. Object/1 applications can query and update data from Oracle through the use of Oracle's SQL language. The Object/1 Professional Pack for Oracle RDBMS is $495.

Micro Data Base Systems, Inc., Two Executive Drive, PO Box 6089, Lafayette, IN 47903-6089, 800.445.MDBS, 317.447.1122, fax: 317.448.6428

IBM IBM **AIX XL C++ Compiler/6000** is generally available and includes a C++ compiler, browser, class libraries, and a test coverage analyzer. XL C++ is a native optimizing compiler that supports the full language definition, including templates and exception handling. Product libraries include I/O stream library, complex mathematics, and task library. The AIX debugger.dbx has been enhanced to support C++ and several AIX commands have been modified to work with XL C++. InterViews 3.0 and NIH 3.0 have been ported to AIX V3.2 to work with XL C++ and are shipped "as is" without support. Prices range from $875 to $7,000 according to the type of processor.

IBM, 1 Kirkwood Blvd., 40-A3-02, Roanoke, TX 76299, 817.961.7326, fax: 817.961.6220

Library Technologies Library Technologies announces the release of **Version 3.0 of C-Heap**, its memory management library for Microsoft and Borland C/C++ compilers. C-Heap now supports the allocation of memory from upper memory blocks (UMBs) through malloc(), either via DOS calls or the XMS driver, transparently. In addition, C-Heap allows utilization of 64K of expanded (EMS) memory as heap space. Version 3.0 of C-Heap adds local heaps to its list of memory tools. C-Heap is $229 including source.

Library Technologies, P.O. Box 56031, Madison, WI 53705-9331, 800.767.4214, 608.274.4224

Borland International Inc. Borland International Inc. announced several products releases. **ObjectVision PRO** is an advanced version of ObjectVision for Windows 2.0. It includes ObjectVision 2.1; Turbo C++ for Windows 3.1; SQL connection, a multimedia tool kit for creating applications with video, sound, graphics, and animation; and **Crystal Reports**, a graphical report writer. ObjectVision 2.1 provides the end user with added database functionality with a new, faster Paradox Engine that supports both 3.5 and Paradox 4.0 file compatibility. Also supported are Paradox MEMO fields, binary large object (BLOB) data types, and composite secondary indexes. Borland **Paradox Engine and Database Framework 3.0** enables programmers to integrate their applications with Paradox data. The new Object Layer provides an object-oriented access layer to all the Engine functions from C++ and Pascal. **Borland C++ & Application Frameworks 3.1 for CD-ROM** simplifies the installation process and includes both uninstalled and pre-installed versions of Borland C++ 3.1. The **Borland KnowledgeBase CD** is Borland's Technical Support Department's database on two CDs. ObjectVision PRO carries a suggested retail price of $495. Borland C++ & Application Frameworks 3.1 for CD-ROM is available to current Borland C++ & Application Frameworks 3.1 users for $19.95 or they may exchange their disk set for the CD free of charge; The CDs can be purchased individually for $249.95. A year's subscription to the quarterly releases is $495. ObjectVision 1.0 users can upgrade to 2.1 for $49.95 while 2.0 users can upgrade to 2.1 for $29.95. The upgrade cost to ObjectVision 1.0 and 2.0 users for the PRO product is $199.95. Current Turbo C++ users may upgrade to ObjectVision PRO for $199.95; ObjectVision SQL users can obtain the product at no cost directly from Borland.

Borland International Inc., 1800 Green Hills Rd., P.O. Box 660001, Scotts Valley, CA 95607-0001, 800.331.0877, 408.461.9000

GUI Computer GUI Computer has a new release of **1.5 ObjectTable C/C++**, a Windows-oriented library that implements a programmable multicolumn table object for the database front end. The new v1.5 release supports protected column, drag the column width at runtime, international currency, different true-type font and color for column, multiline column title, 3D column title, and row title. It is compatible with Borland Resource Workshop and Microsoft Dialog Editor, and supports Borland ObjectWindows C++ and Microsoft Foundation Class C++. For C++ , 1.5 ObjectTable C/C++ is priced at $99 for object code and $259 with source; for C, $79 for object code and $199 with source.

GUI Computer Inc., PO Box 795908, Dallas, TX 75379, 214.250.3472, fax: 214.250.1355

Product Announcements

Product Announcements is a service to our readers. It is neither a recommendation nor an endorsement of any product discussed.

Pocket Soft

Pocket Soft unveiled the initial release of a new suite of memory management libraries, virtual-memory data (VMData), providing a uniform, cross-platform method of managing a program's dynamically-allocated data. VMData provides a platform-independent virtual-memory scheme for managing dynamically allocated data, distinguishing between different types of memory and managing program data according to data priority. VMData accesses all the available memory resources for DOS, Windows, and OS/2, such as EMS, XMS, UMB, HMA, Windows movable, etc. The VMData libraries are for C and C++ languages. The supported operating systems are MS-DOS 2.1 and above, Windows 3.C, and OS/2.1.X and 2.0. VMData costs $495 for the first platform purchased and $295 for each subsequent platform.

Pocket Soft, Inc., P.O. Box 821049, Houston, TX 77282, 713.460.5600, 800.826.8086, fax: 713.460.2651

Virtual Technologies

Virtual Technologies Incorporated announced the commercial launch of the SENTINEL debugging environment: a comprehensive debugging tool supporting runtime verification of pointer usage and dynamic memory allocation in both C and C++. SENTINEL is supported for Solaris 1.0.1 (SUN OS 4.1.2) on SUN SPARC-II and compatibles, HP-UX 8.0 on HP 9000/8xx systems, and System V Release 3.2 and 4.0 on Intel 80386 and 80484 systems. The environment is priced at $195 in Intel 80x86 environments, $395 in SUN environments, and $495 in HP environments. Substantial discounts apply for multiple unit purchase.

Virtual Technologies Incorporated, 46030 Manekin Plaza, Suite 160, Sterling, VA 22170, 703.430.9247, fax: 703.450.4560

Knowledge Garden Inc.

Knowledge Garden Inc. announced KPWin++, which allows users of its KnowledgePro for Windows (KPWin) high-level object-oriented development environment to generate C++ code. Users of Revelation Technologies Inc.'s database tool for Windows, OpenInsight, which includes a licensed copy of KPWin, are also able to take advantage of the new C++ code-generation tool. KPWin++ reads code written in KPWin or OpenInsight environments and generates ANSI-standard compilable C++ code. Using the Microsoft C/C++ 7.0 compiler, users are able to amend generated C++ code, link to third-party libraries if needed, and then compile to create an executable file. Later implementations will also support the Borland, Watcom, and Zortech compilers. KPWin++ is priced at $895 or at a $695 introductory price to KPWin and OpenInsight users. The expert system and hypertext DLL engine used by KPWin++ will also be available for licensing and embedding into other tools and applications.

Knowledge Garden Inc., 12-8 Technology Drive, Setauket, New York 11733, 516.246.5400, fax: 516.246.5452

Objectivity Inc.

Objectivity Inc. has developed a localized version of Objectivity/DB, its object database management system, for the Japanese market. The localized version of Objectivity/DB allows users to store, retrieve, manipulate, and display multibyte Kanji characters. All Objectivity application development tools are fully integrated into the Japanese operating environment and support Kanji messages, menus, and online help. Objectivity provides Japanese user documentation and works with its codevelopers/distributors, Mitsui and Osaka Gas Information System Research Institute (OGIS-RI), to conduct Japanese seminars and training and provide local support in Japan. Support for Chinese and Korean characters will be provided by the first quarter of 1993. **Objectivity Inc., 800 El Camino Real, Menlo Park, CA 94025, 415.688.8000**

Pioneer Software

Pioneer Software and Borland International have announced the release of Q+E DataLink/OV, which enables ObjectVision developers to link their applications to ten additional major database formats. Q+E DataLink/OV provides database access via a set of self-registering ObjectVision functions that perform database operations. With this new product, ObjectVision users can augment ObjectVision's database access with SQL data and other major data formats. Q+E DataLink/OV has a suggested retail price of $399. Pioneer Software is offering Q+E DataLink/OV for a special introductory price of $299 until January 1, 1993.

Pioneer Software, 5540 Centerview Drive, Suite 324, Raleigh, NC 27606, 919.859.2220, Fax: 919.859.9334

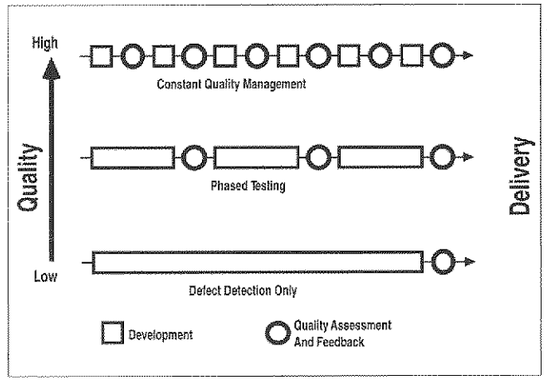


Figure 2. The evolution toward constant quality management.

design, or implementation of a system is when the change occurs. This requires a methodology that supports fully integrated, distributed testing at all points in the lifecycle. In the early years of software development, the user's reaction to the product upon delivery was the only kind of quality management. And while large-grain, phased-testing approaches have been used in traditional software development for years, they have not been able to deliver much more than the "zero defect" level of quality. The lesson to be learned is: Decrease cycle time between construction and evaluation and you increase quality. The logical result of this evolution is a complete distribution of testing and quality assessment across all activities in the lifecycle (Figure 2). That's what constant quality management is all about.

An important consequence of this approach is that for any significant system the level of detailed management required to fully implement constant quality management demands simultaneous support by both automated tools and development methodology. Ed Yourdon puts it this way in his latest book, THE DECLINE AND FALL OF THE AMERICAN PROGRAMMER:

> . . . the situation just described suggests that something more fundamental is going on: software development methodologies are created (by someone, or by some motley crew of people) and introduced into the field. If they survive, inevitably they evolve over a period of time. Meanwhile CASE tools evolve too—but the key point is that they may lag behind the methodologies (by several years, in the case of structured techniques!) until the CASE tools themselves become the driving force for methodology creation and evolution.[7]

This assertion has been a top requirement for our work on both methodology and its supporting environment. In the following sections, we will describe how each specifically supports constant quality management.

Methodology support

So how does an organization go about implementing this approach to quality management in its development activities? Zultner advises:

More attention needs to be given to finding out what features are expected (and thus not mentioned), and which are unimagined (and thus not mentioned) but desired—and delightful when delivered. Substantially more effort must be spent up front during analysis and design to catch defects soon after they're made. Inspections—or rigorous (structured) walkthroughs—should be much more frequent during analysis and design.[4]

Closed-loop control and reducing cycle time are process issues, not technology issues. Therefore, the methodology that guides the process of development must fully support distributed testing and continual quality feedback. Unfortunately, most O-O methodologies today simply suggest that we "iterate until we're satisfied," giving little if any process guidelines for evaluation and certification of requirements, design, or implementation. That is one reason we were driven to develop a new methodology focused on the entire development lifecycle.[8]

> ***Just as no modern organization can function with manually maintained paper accounting ledgers, no software development organization can successfully manage this amount of information without automated tools.***

By defining the responsibilities (requirements) for the entire system as well as its components, and by focusing on the interaction among the behavioral entities in the system, the same conceptual model of interacting behavioral entities can be applied at all levels of granularity in the design. Then, by using the scenarios defined for each responsibility, the fitness of a proposed design or implementation change can be tested by roleplaying (for high-level component interactions) or by developing software test harnesses (for method-level services). Since every entity in the design has responsibilities, whether it be the entire system, a component (a group of classes), or an individual object class, testing and other quality assessment can occur at all levels of the design at any time in the process.

How much testing is done at any one time can then be determined by management using return-on-investment principles to justify the effort required for expected increase in total system quality. And because the behavior of any system part can be represented using CRC cards, users and domain experts can be brought in at any time to roleplay changes in system behavior and provide external validation of system requirements and design.

Tool and environment support

Constant quality management requires a much finer level of monitoring and control of the entire process of developing software, and therefore produces a much larger quantity of valuable information that must be managed. Just as no modern organization can function with manually maintained paper accounting ledgers, no software development organization can successfully manage this amount of information without automated tools. But the tools must be in sync with the methodology or the mismatch will cause more problems than it solves. In our previous article we defined tool requirements for the successful creation and management of high-quality software assets.[9] With respect to testing via scenarios, all scenario information as well as roleplaying results must be managed in the same multiuser, version-controlled environment as the rest of the design information. Automatic management of lifecycle information from requirements to code and back creates an opportunity to dramatically increase awareness of both designers and programmers of the effects their decisions have on system quality: real-time metric feedback.

Metrics support

"You can't manage what you can't measure." This old business adage has never been more true than when applied to software development. With so many different kinds of activities going on throughout the lifecycle, how can project managers effectively track the progress of the project, let alone the level of quality being produced? The truth is they can't, at least not alone. Remember, developing high-quality software requires that everyone in the process be responsible and empowered to positively affect that quality. Programmers as well as managers need feedback; the closer the feedback to the system addition or change, the faster the cycle time and the higher the quality of result. If you change a service (method) in a class you should be immediately notified as to the completeness of its interface signature, the complexity of your algorithm, the readability of your code, etc. Remove a collaborator from the CRC card for a class, and you should be immediately aware of the methods now invalid because of their coupling to the removed object's services.

This level of evaluation and feedback has been a requirement in the lifecycle development environment supporting our methodology, and for the past six months has had a major impact on the quality we have come to expect in the continuing development of the tools themselves. Given these benefits, we have dedicated additional resources to develop more and better ways to measure software quality in all its expressions and provide more effective feedback in the environment. We have also recognized the great potential for developing project management extensions, such as schedule management and time estimation to the environment, based on the same kinds of metrics when viewed from a multiuser, project-wide perspective.

MAXIMIZING RETURN ON INVESTMENT IN OT

Obviously, a long-term commitment is required by the highest levels of corporate management to refocus an entire software organization around these quality ideals. The transition will not be easy or inexpensive, but it is an absolute necessity if the organization is to achieve the level of software quality required for corporate survival in the decades to come.

Each of the four articles in this series has focused on an essential requirement for meeting the challenges of enterprise computing and maximizing an organization's return on its investment in object technology:

1. Software must be treated as an asset worthy of investment.
2. A lifecycle methodology must be adopted that unifies and supports the development process from requirements to code and back.
3. An integrated multiuser, version-controlled environment for the creation and management of software assets must be deployed, fully supporting every aspect of the lifecycle methodology.
4. Constant quality management must be practiced throughout the lifecycle and fully supported by the methodology and environment.

All these requirements are interdependent and must be met for success. At KSC, we are actively seeking partnerships with those organizations ready to begin the journey. ≡≡

REFERENCES

1. Adams, S. Return on investment: Software assets and the CRC technique, HOTLINE ON OBJECT-ORIENTED TECHNOLOGY 3(10):4–7, 1992.
2. No. 1–and trying harder, BUSINESS WEEK, Special Issue on Quality, October 1991.
3. A new era for auto quality, BUSINESS WEEK, October 22, 1990.
4. Zultner, R. THE DEMING WAY TO SOFTWARE QUALITY, presented at the Pacific Northwest Software Quality Conference, Zultner & Company, Princeton, New Jersey, 1989.
5. Adapted from Kano. Exciting quality, HINSHITSU 14(2), 1984.
6. Anderson, D.M. Time warrior, Success, December 1991.
7. Yourdon, E. THE DECLINE AND FALL OF THE AMERICAN PROGRAMMER, Yourdon Press, New York,1992, p. 32.
8. Adams, S. Return on investment: extending CRC across the lifecycle, HOTLINE ON OBJECT-ORIENTED TECHNOLOGY 3(11):6–10, 1992.
9. Adams, S., Return on investment: development environments for the lifecycle, HOTLINE ON OBJECT-ORIENTED TECHNOLOGY 3(12):1, 7–9, 1992.

Sam Adams is the Senior Consultant and cofounder of Knowledge Systems Corporation. Since 1984, Mr. Adams has been actively developing object-oriented software systems in Smalltalk and is widely recognized for his expertise. He is codeveloper of the group facilitation technique using CRC cards and has been training computer professionals in object-oriented technology for over six years. Mr. Adams has served on several conference committees and is a frequent speaker and panelist at leading industry conferences. He can be reached by phone at 919.481.4000 or by fax at 919.460.9044.

Comparison of OOSE with other object-oriented methods

The chapter on other object-oriented methods discusses, on a survey level, other object-oriented development methods. Jacobson notes that comparing how methods help achieve a better and more competitive product in terms of quality, productivity, modifiability, etc., would be most interesting, but comparing notation, concepts, and easy-to define strategies is much more achievable. He chooses to provide the latter and offers an overview of each of the following methods followed by a notational comparison:

- OOA by Coad-Yourdon
- OOD by Booch
- HOOD
- OMT by Rumbaugh
- CRC by Wirfs-Brock

Jacobson refers to Rebecca Wirfs-Brock's method as RDD (requirements-driven design), according to Wirfs-Brock's own terminology. Most people use CRC to identify her method, but RDD is, in fact, a registered mark of a San Jose–based systems engineering company, Ascent Logic.

TEST CASES

Most object-oriented books provide examples based on the preexistence of requirements. Typically, they give detailed examples of how a particular architecture and classes resulted from requirements A and B.

Interestingly enough, object-oriented developers find categorizing and grouping requirements very difficult. Jacobson spends less time describing content and more time describing the pros and cons of potential approaches. Examples of this abound in Chapters 13 and 14. In Chapter 13, Jacobson discusses the pros and cons of describing a particular sequence as a single, complete use case vs. dividing it into several. Another example in the same chapter is a discussion of options for storing attribute information during the analysis phase.

SUMMARY

This very interesting book on object-oriented software engineering would be an asset to your personal library. As I mentioned above, the preface provides a "roadmap" of which chapters to read. Anyone unfamiliar with Jacobson's concept of a use case should first read the explanation in Chapter 7 and then return to the sequence in the preface.

For those who remember my "Research or Ready" article,[2] reading Jacobson's book was the first step in the review process for inclusion in my list. I'll include more details as I complete the review.

As a final note, don't overlook Dave Thomas' well-written preface. ≡≡

References

1. Jacobson, I. OBJECT-ORIENTED SOFTWARE ENGINEERING—A USE CASE DRIVEN APPROACH, Addison-Wesley, Reading, MA, 1992.
2. Dock, P. Research or ready? HOTLINE ON OBJECT-ORIENTED TECHNOLOGY 3(9):1, 7–9, 1992.

Patti Dock has been involved in the object-oriented marketplace since1985 when she joined Stepstone as a technology consultant. After leaving Stepstone, Patti worked for Jackson Systems Corporation and General Electric, both companies actively involved in object technology. Today Patti is Vice President at OrgWare, Inc., where she consults with organizations as they migrate to object-oriented techniques. She also teaches a course called OBJECTMethods, which compares and contrasts leading object-oriented design methods. Patti can be reached at 203.270.1242.

SOFTWARE ENGINEERING ASPECTS

There is no shortage of object-oriented design and analysis books on the market today. This book, however, is one of the few that addresses object-oriented software engineering topics, such as reuse through components, testing, and project management.

Components

Jacobson states, "To build with components and to build with objects are two entirely different activities." The chapter on components discusses one of the difficult problems in the object-oriented market today: Why can't we seem to produce reusable components? He lists seven of the most commonly cited reasons why we have failed, and suggests that to successfully create components we need both a methodology and organization that support their lifecycles. These reasons are:

1. Project schedules and budgets do not allow for the time required to develop quality components.
2. The not-invented-here attitude precludes reuse of someone else's design or code.
3. There is no recognized standard for component functionality and use.
4. Components that exist cannot be found because our libraries and tools are not built around component reuse.
5. Measurement of lines of code defeats the motivation to reuse (i.e., reuse is not doing work, while writing code is doing work.)
6. Payment schemes (i.e., pay-per-use) for a component marketplace have not been implemented, so the economics of component distribution are unfavorable.
7. Engineers have tried to define components from functions, not objects.

He discusses methodology support and identifies many of the organizational issues associated with maximizing reusability.

Testing

Jacobson devotes an entire chapter to the neglected issues of quality assurance and testing. He mixes basic testing philosophy with a discussion of special testing challenges introduced by polymorphism and inheritance. (Portions of this chapter may read like the testing chapter of your college software engineering course, but I think that's okay since most of us didn't pay enough attention to that chapter anyway!) He emphasizes the importance of use cases in designing your testing procedures and reiterates the need for automation of regression testing in an object-oriented system.

Project management issues

The "Managing object-oriented software engineering" chapter discusses some of the experiences gained by introducing ObjectOry to about 15 real projects of varying size (3–50 man years) during the past few years. This chapter discusses and describes:

- how introducing any new process is delicate and should be only undertaken when the process is in sync with the corporation's long-term plan for development organization
- how the objectives of organizational change should be clearly understood and supported
- how to select a pilot project and the importance of carefully tracking its progress

Prototyping must be integrated and managed and the defined goal should be a higher-quality result. Real experiences with project staffing, project tracking, training, risk analysis, project management, project organization, and project metrics are also discussed.

SPECIAL TOPICS

In addition to software engineering topics, this book contains some interesting perspectives on a number of special topics such as database specification, real-time experiences, and a comparison of existing object-oriented design methodologies.

Database specification

Jacobson's chapter on database specification provides a clean description of the differences between relational and object-oriented databases complemented by an explanation of when and why system architects must make choices.

The chapter describes how to encapsulate some of the problems that arise when one uses a relational database for persistent object storage. A reusable framework is used to create the logical database design from the object-oriented model. It simulates the inheritance structure using tables.

Jacobson believes there is no industry consensus today on describing an object-oriented database management system (ODBMS). He further points out that database design must be integrated during analysis and design of the application because today's commercial ODBMSs use object-oriented implementation languages such as C++ or Smalltalk rather than a specific data-manipulation language (such as SQL) like relational databases.

Real-time specialization

Jacobson defines a hard real-time system as one in which timing constraints must be met to avoid catastrophes. Real-time requirements add time as an extra dimension to system design. Process synchronization and communication almost always have a major impact on design criteria.

During the analysis process, one must collect important real-time constraints and document them in use case descriptions. During construction, the system must be implemented and tested against these constraints. Use cases are an effective tool to document system behavior and real-time behavior. Distribution of objects across several processors can be encapsulated in the object itself. Interaction diagrams are used to handle real-time requirements and allocate timing requirements for different objects. Jacobson explains why testing is probably the most critical aspect of real-time systems. He contends that requirements traceability coupled with automated regression and use case testing are crucial to the high quality demanded in hard real-time systems.

Evolving markets for software components

Howard Baetjer William Tulloh

As industry luminaries such as Brad Cox[1] and Fred Brooks[2] have noted, the development of robust markets in software components (meaning systematic, widespread, routine commerce in reusable components) promises to provide great improvements in software productivity. Markets distribute the cost and complexity of software development across many organizations, allowing firms to focus internal capabilities on their own competitive advantage while purchasing external capabilities on the market.

In a vigorous software component market, developers could routinely purchase much of the functionality to be used in their applications in the form of components routinely developed by specialists building such components for sale in the market.

Such markets, long anticipated by proponents of object-oriented technology, have been disappointingly late in coming. But the wait may end soon. Pieces of the puzzle are increasingly falling into place and there is now an electronic software component marketplace—an electronic "location" where buying and selling components is inexpensive and convenient—in which robust component markets may emerge. This column synthesizes some of our research in trying to overcome barriers to the software component markets on the American Information Exchange (AMiX). We explain how electronic marketplaces, such as those on AMiX, can help to catalyze extensive commerce in software components.

CHALLENGES TO COMPONENT MARKETS

For robust component markets to develop, a number of challenges must be met:

- **Developing new channels of distribution.** Current software distribution channels, generally expensive and aimed at the mass end-user market, are ill-suited to components, which require inexpensive distribution channels aimed at developers and sophisticated end users. The industry needs to develop affordable means by which producers can easily distribute their components and users can easily access them.
- **Making components understandable.** The conceptual nature of software requires that extra attention be paid to making components easy to reuse. Component producers need to provide guidance for users of those components, making clear the contexts in which components may be built into larger applications.
- **Assuring quality components for sale.** Components must undergo the same level of quality assurance (testing, documentation, and availability of support) that applications undergo (or should undergo) if they are to become widely marketable. Component producers must come to see their reusable components not as by-products of application development, to be reused if possible, but as important products in their own right, deserving of significant time and effort to prepare them for market.
- **Creating standards for component interoperability.** A major problem with class libraries is that they are often incompatible with other class libraries, even those built in the same language. Accordingly, the potential component market is highly fragmented. Industry standards for component interoperation are needed. As interoperation becomes less costly, the number of potential buyers for any given component will increase. Bright spots in this respect are OMG's CORBA, IBM's System Object Model (SOM), and Digitalk's PARTS.
- **Addressing ownership and liability concerns.** The software industry faces a serious challenge in ensuring adequate compensation to developers. This challenge is greater still for components, which are both finer-grained than applications and also intended for multiple use in various applications. At the same time, liability issues arise: Who is responsible when a component does not work as expected? The industry needs new means of licensing components that clearly define ownership and liability.

FROM ELECTRONIC DISTRIBUTION...

A promising point of leverage for overcoming these challenges is electronic distribution of components. This is a natural match because electronic distribution provides an affordable means of access to components. As networking and distributed computing become more ubiquitous, component distribution inevitably will be electronic.

But electronic delivery by itself does not allow a component user to obtain usable components at reasonable cost. Even if the world's best components were made available on a universally accessible network, users probably would not find it affordable to use them. Although the dollar cost of downloading appropriate components would be negligible, the cost in time and effort

would be very high. Progress is being made in developing better search tools but it is still very costly to search and retrieve from a passive repository of components.

...TO ELECTRONIC MARKETS

Electronic markets go far beyond electronic distribution because they are active rather than passive. The relationship of electronic distribution to electronic markets is the same as a large, partitioned building surrounded by parking lots to a shopping mall. The electronic network provides the same supporting infrastructure—the virtual location where component commerce can take place—as the large, partitioned building provides for retail trade. But neither is a market, nor can we make them markets by filling them with priced goods. Picture a shopping mall building with no signs, no salespeople, no displays, few shelves or racks, no brand names, no segmentation according to expense—nothing but lots of merchandise grouped by category. That's not a market, just as plain electronic distribution is not. If you know what you want and where to find it (or you have extraordinarily capable search tools), you can get what you want without difficulty. Otherwise, forget it.

The point of markets is to make it easy for buyers and sellers to find one another and exchange successfully. What markets provide beyond simple technical availability of goods is rich interaction among market participants and, consequently, a wealth of information, guidance, and evolving practices and institutions. Markets offer the active participation of vendors who achieve their purposes only by providing customers what they want and need at a reasonable price.

THE AMERICAN INFORMATION EXCHANGE: A FIRST EFFORT

The American Information Exchange Corp. (AMiX) provides an electronic marketplace for software components. AMiX provides basic institutions and capabilities for electronic markets for information of all kinds, and ultimately plans to open a wide variety of information markets. In its early stages, however (the system came online the first quarter of 1992), AMiX is building markets in computer-related areas, including markets for software components and consulting. The first to be developed is the Smalltalk market. (There is also a C++ market; AMiX plans to offer a variety of other component markets in the near future.) The Smalltalk community has been enthusiastically supportive of the concept. Smalltalk users program by extension, using what already exists in their environment to support their problem solving. Programmers typically take advantage of the installed base of class libraries by browsing class hierarchies for what they need, editing and subclassing as necessary. The AMiX Smalltalk market is essentially an extension of the Smalltalk paradigm.

A solution to the distribution challenge

The primary benefit of the AMiX service for fostering component markets is its low-cost distribution system. Components can be inexpensively stored on the system and downloaded by buyers for immediate use. To facilitate custom development, the system supports small-scale consulting with negotiation, contracting, and delivery online. AMiX handles all billing and accounting centrally, freeing market participants from accounting overhead. Online charges are at cost and, in any case, the system allows users to do most of what they need to do from their local image of the system, connecting only for short periods. AMiX profits only when market participants conclude a mutually satisfactory exchange, taking a percentage of the purchase price.

Market-driven answers to the remaining challenges?

Of the challenges mentioned previously, AMiX solves the first directly. AMiX is committed to being an open marketplace where all participants may be buyers, sellers, or both, as long as they maintain basic standards of good citizenship. Accordingly, AMiX individual vendors are responsible for the quality, understandability, and licensing arrangements of their components, and for the components' adherence or non-adherence to standards.

We believe AMiX can catalyze answers to these remaining challenges by providing a means for customers to express their needs in a competitive market context. These challenges must be met through the actions of various players in the software industry, whether vendors or users. AMiX provides the electronic "location" and system support for real market activity. Within the context of actual evolving AMiX markets, these challenges be can met in a coherent fashion.

The AMiX Smalltalk market provides:

- a rich information environment. Through extensive system hyperlinking, users can easily access a wealth of supporting information to evaluate vendor reputations and product quality. These include sellers' resumes (participants are required to publish a resume before they may publish or sell anything on the system), buyers' evaluations of products and services, component reviews and recommendations, etc.
- a competitive context, with corresponding market pressure on component producers to reduce prices, improve interoperability, and steadily enhance and improve their components. It is easy to compare products on the system; all prices and terms are immediately visible.
- active matching of vendors' expertise and customer needs. Component producers actively market their components according to research users' needs, bringing the product to the users' attention rather than leaving it in a repository. Conversely, component users bring their needs to component producers, bidding for the (custom) construction of needed components, support, training, and other services. Buyers and sellers can negotiate and establish binding contracts, deliver, and pay for services rendered—or do any other business—online.

Understandability and quality beyond "take-it-or-leave-it"

Within this market setting, there is pressure from customers for more understandability and quality assurance. Potential users of

Reviewing OOSE: a use case driven approach

Patti Dock

I recently read OBJECT-ORIENTED SOFTWARE ENGINEERING—A USE CASE DRIVEN APPROACH by Ivar Jacobson,[1] a good object-oriented design and analysis book and an excellent object-oriented software engineering book. Not only is the technical content high, but I can honestly say I enjoyed reading it. It captures 20 years of the author's experience and is written in a very modularized fashion. The preface even provides a "roadmap" of which chapters to read depending upon one's experience level and interests.

The "Introduction" section's five chapters define new terms and explain OOSE's concepts, approach, and lifecycle. They introduce ObjectOry (the Object Factory for Software Development), the development technique used in this approach.

The next seven chapters, comprising the "Concepts" section, provide the core of the analysis and design method. This section introduces the use case concept and explains its role in design, analysis, and testing. There are also individual chapters on realtime specialization, database specialization, components, and testing.

The "Application" section contains two case studies augmented by two chapters, one addressing the managerial issues associated with object-oriented projects and another, very interesting chapter comparing OOSE with five other object-oriented analysis and design methods.

USE CASE DRIVEN APPROACH

Throughout the book, Jacobson uses OOSE techniques to describe OOSE. The title of the book is derived from a concept Jacobson has been using and documenting over the past decade. Actors and use cases are used to define what exists outside the system. An actor represents a role a user may play. The user interacts with the system by performing a sequence of transactions, referred to as a use case. That is, from a user's perspective, each use case is a complete course of events in the system. Examples of use cases might be an operator connecting a subscriber to a system, an insurance adjuster entering an estimate, or a reservation clerk verifying the availability of a rental car.

Jacobson continually emphasizes the importance of viewing the system design process from an O-O perspective. Each use case is a class. Individual instances of use cases are the class (or objects). This allows us to view each use case instance as a transaction with internal states.

A use case driven design is achieved when each use case model is described by a number of actors and use cases, and each use case has detailed descriptions and interfaces. When the system is in operation, instances are created from the descriptions in this model. To modify the system, we simply remodel appropriate actors and use cases.

OOSE CONCEPTS

This section of the book concentrates on incremental and creative activity to achieve the following five models, resulting in a completed system:

- a requirements model that captures functional requirements for the system
- an analysis model that provides a robust and modifiable object structure
- a design model that adopts and refines object structure to the current implementation environment
- an implementation model that describes implementation of the system
- a test model that concentrates on verifying the system

OOSE describes how each of the models look (syntax), what each means (semantics), and the appropriate heuristics and rules of thumb (pragmatics.)

OOSE describes two processes: analysis and construction. The analysis process produces both requirements and analysis models. Use cases are used in the requirements model to describe the functionality of the system. These use cases provide the foundation for analysis, implementation, and test models. The requirements model itself provides the basis for the analysis model. The analysis model specifies all the logical objects to be included in the system and how they are related and grouped. The design, implementation, and test models are completed during the construction process.

> *Our attitude towards software should be the same as toward a security investment. We are going to purchase the software, so we should think first about the ROI we expect.*

- The processing required by a distributed application can be spread across machines, resulting in better capacity utilization.

However, none of these advantages will provide ROI in the abstract. Possible cost increases associated with adopting a distributed information system can immediately be cited:

- Many more machines must be purchased, installed, connected, and maintained.
- Software must be purchased for all machines that need it and must be upgraded in a coordinated manner to avoid inconsistencies.
- Problems may occur in many more machines, making them harder to isolate and correct.

CONCLUSION

It is evident that without more specific information about a business, it is impossible to evaluate whether purchasing software for distributed information systems will generate a positive ROI. In the next series of articles, I will develop scenarios for ROI evaluation, from which guidelines for conducting evaluations can be articulated based on ROI analysis. This will help both purchasers and vendors identify the information needed to create successful partnerships yielding distributed-information systems that provide significant ROI. ≡≡

Tim Andrews has been Chief Technical Officer at ONTOS Inc. since 1988. He is one of ONTOS' primary designers and has a background in object technology, database implementation, and technical marketing. Mr Andrews has helped shape the strategic direction of the ONTOS product architecture and has continued his work with critical customers, most notably IBM. He can be reached at ONTOS, Three Burlington Woods, Burlington, MA 01803, by phone at 617.272.7100, or by fax at 617.272.8101.

CALENDAR

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: Robert Shelton, 1850 Union Street, Suite 1584, San Francisco, CA 94123; fax: (415) 928-3036.

November 5, 1992 Knowledgeware User's Group and DAMA (NY Chapter) New York, NY Contact: 212.439.0063	Nov. 16–20, 1992 C++ World Meadowlands Hilton, NJ Contact: 212.274.9135	February 1–4 and February 4–5, 1993 OOP '93 and C++ World Munich, Germany Contact: 212.274.9135	April 19–23,1993 Object Expo New York, NY Contact: 212.274.9135	February 1–4, 1993 Object World (OMG) Boston, MA Contact: 800.225.4698
March 8–11, 1993 X World New York, NY 212.274.915	March 8–12, 1993 INTEROP Washington, DC 800.INTEROP	March 17–19, 1993 Uniforum '93 San Francisco, CA 800.323.5155	May 3–7, 1993 DB Expo San Francisco, CA 415.966.8440	June 14–17 Object World SF San Francisco, CA 800.255.4698

components have been frequently blamed for not using available components due to an "not invented here" (NIH) attitude. But as Ward Cunningham and Kent Beck point out,[3] this phenomenon is probably more often the result of the take-it-or-leave-it attitude of component suppliers who are not offering understandable, tested code. In a competitive market setting such as AMiX provides, with immediate electronic access to suppliers, customers take an active role in demanding high-quality, easily understood components. When buying reusable components, they expect to buy not only code but documentation, test suites, and support.

As the market grows, however, customers may not need to take so much initiative. Their demand for quality and reliability may generate the emergence of supporting institutions and organizations such as cataloguing services, testing services (similar to Underwriters' Laboratory), publications that review and/or rate new products, comparative testing services like CONSUMER REPORTS, etc.

DRIVING LICENSING AND STANDARDIZATION ISSUES

By reducing the transaction costs of bringing together vendors and customers, electronic markets provide a complementization to formal standardization processes with regard to component interoperability and licensing. Electronic markets help identify key areas of concern and provide direct profit incentives for their removal.

CONCLUSION

The challenges facing software component markets will be met in an evolutionary fashion through the efforts of many people and organizations. Robust component markets will evolve as progress in one area increases the incentive to progress in another. The availability of a new electronic marketplace for components may be a catalyst for meeting remaining challenges by bringing component users and producers together in a competitive business environment. Facing competitive market selection for the most usable and dependable products, producers will be driven to improve the quality and understandability of their components. In seeking access to the broadest possible range of products, component users will demand faster adoption of interoperability standards. The emergence of these standards, in turn, will stimulate a burst of component development. As revenue streams grow, producers and suppliers will be driven to develop successful licensing arrangements to protect their interests. In such a complex, interactive manner, we can expect component markets to grow. ≡≡

References

1. Cox, B. Planning the software industrial revolution, IEEE SOFTWARE, November, 1990.
2. Brooks, F.P. Jr. No silver bullet: Essence vs. accidents of software engineering, COMPUTER, April, 1987.
3. Cunningham, W., and K. Beck. Constructing abstractions for object-oriented programming, JOURNAL OF OBJECT-ORIENTED PROGRAMMING 2(2):17–19, 1989.

The authors are researchers with the Agorics Project at the Center for the Study of Market Processes at George Mason University, where Mr. Baetjer is Executive Director and Mr. Tulloh is Research Coordinator. In addition, Baetjer and Tulloh are principals in Agoric Enterprises, Inc., helping to build software components and consulting markets through the American Information Exchange Corp. (AMiX). They can be reached at Agoric Enterprises, 10364 Bridgetown Pl., Burke, PA 22015, by phone at 703.250.4760, or fax at 703.250.3532.

SOFTWARE ENVIRONMENT FOR THE NINETIES *continued from page 4*

viding them with a common, object-oriented interface. This interface allows an existing, non–object-oriented application to behave in an object-oriented manner by correctly accepting and responding to other objects on the network. This approach allows a smooth transition of non–object-oriented programs into the object-oriented environment.

Object-oriented technology has tremendous implications for engineering application developers. No longer do they have to concern themselves with specific details of a network—locations and types of plotters, numerically controlled (N/C) machine tools, computers, databases, etc. Applications can automatically adjust to the changing manufacturing environment. For example, assume a shop-floor scheduling application needs to query a machine tool about the number of parts completed since its last maintenance. To accomplish this data exchange today, a system developer must hardwire the communications link from the machine tool directly into the computer running the scheduling application. The result is a static, single-use gateway between two individual resources.

Under this scenario, a problem will arise when the location of the machine tool is changed or a new machine tool controller is added. When this type of change occurs, the link between systems must be redefined and the scheduling application—along with every other application communicating with the machine tool—must be altered.

In a distributed object-oriented system, a much greater degree of flexibility is achieved. The machine-tool and the scheduling application are defined simply as objects on the network. Objects interact with each other by passing requests and responses and no object in a transaction needs to know location or implementation details. In an object-oriented system, changes to a machine tool's location or controller has no bearing on other objects on the network.

Because of their flexibility, object-oriented systems are obviously less confusing to the network user. On most systems today, a user who wants to print an E-size drawing, for example, needs to know how to route the drawing to a specific plotter. In an object-oriented system, both drawing and plotter are objects and the network is aware of the requirements and capabilities of each. The user merely selects Plot to generate a drawing, and the system will forward it to a plotter with the appropriate characteristics.

The use of a graphical interface offers the user even more flexibility by presenting objects as icons. By "dragging" the icon representing his drawing and "dropping" it over the plotter icon, our user has accomplished the same task as before. The drawing may be of any size, and the plotter may have any address on the

network. The user can get on with the job at hand. The object management system handles the details.

ORB: THE OBJECT MIDDLEMAN

The heart of the distributed object management system is the object request broker (ORB). Like any broker, the ORB acts as a middleman to help two entities arrange a transaction more easily and efficiently than they could alone.

In a distributed environment, objects turn to the ORB to set up interactions. For example, an application requesting printing needs to provide only two pieces of information: the name of the operation it wants to perform (in this case Print) and the information that needs to be printed. It doesn't have to know the printer's location on the network, format requirements, or communications protocol. The ORB takes the application's basic request, locates the object that can perform the service, fills in the details needed to request the service, and communicates with the targeted object. The ORB then returns to the application the result of the requested operation.

THE NEED FOR STANDARDS

The full benefit of distributed object management systems in the manufacturing industry won't be realized until core data format and communications standards are established. With such standards, users will be able to mix and match best-of-class applications and other objects as needed to create a custom-integrated, distributed solution to their scientific and business systems requirements.

The Object Management Group has laid the groundwork for interoperability of applications across heterogeneous networks with its Common Object Request Broker Architecture (CORBA). Backed by 180 members—including Digital Equipment Corp., Hewlett-Packard, SunSoft, NCR, Hyperdesk, and Object Design—CORBA provides a standard interface for developing distributed management system applications, allowing objects to interact across a network. But while CORBA specifies the mechanism for achieving this interaction, it provides wide latitude in defining how the objects on the network are actually implemented. Each implementor, for example, is free to create a unique ORB core.

Meanwhile, the International Standards Organization (ISO) has specified a standard data format specifically geared to the needs of the manufacturing industry. Standard for the Exchange of Product Model Data (STEP) enables everyone involved in the design, manufacture, and support of a product to create, access, and share information. STEP is vendor neutral and is capable of completely and accurately representing data throughout a product's development and production cycles. The accuracy of the STEP representation makes it suitable for neutral file exchanges, as well as a basis for implementing and sharing databases, enabling users running multiple applications to access the data simultaneously.

A STEP model includes such particulars as product information (part number, version, security classification), tolerance specifications, material specifications, surface-finish information, feature definitions, shape (both geometry and topology), and more. Collectively, this information completely defines the physical and functional characteristics of a component (see sidebar).

The STEP standard is able to support a product throughout its development and production cycles. Design specifications, analysis results, test results, manufacturing process plans, setup sheets, tooling and numerical control data, quality assurance inspection information, and support data are all captured as a product passes through these stages of its lifecycle.

Finally, STEP solves the problem of incomplete or inconsistent data exchange by rigorously defining both the information required to specify a component and the constraints that apply to this information. To do this, STEP uses a formal information modeling language called EXPRESS to define schema for data that is to be captured and exchanged. In addition, any constraints on the data are also captured in the EXPRESS-based information model. This unambiguous definition leaves no room for vendors to misinterpret the standard and ensures that implementations can be rigorously tested to verify their compliance.

THE WINNING COMBINATION

An object-oriented distributed computing environment coupled with an industry-standard model utilizing both CORBA and STEP specifications could fundamentally change the way manufacturing companies do business by eliminating many of today's bottlenecks and inefficiencies. In such an environment, object software can be easily plugged into the network, offering several distinct benefits:

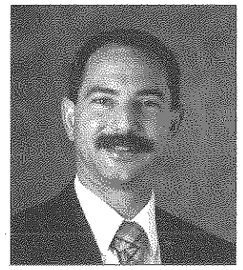
- Customers can employ the latest best-of-class applications without abandoning valuable, existing data.
- Integration of existing applications is simplified.
- All applications can be linked by a single standard architecture rather than customized, fixed linkages between individual applications.
- Object-oriented systems have the potential for tremendous gains in productivity, as applications are able to seamlessly interact, sharing common data.

Object-oriented technology combined with industry standards has the potential to deliver enormous returns to the manufacturing industry. The improved functionality, flexibility, and reliability offered by object-oriented systems will offer manufacturers the competitive tools they need to shorten time-to-market while maintaining a consistently high level of quality. The next industrial revolution will thus provide the same benefits as the original, at a time in history when global competitiveness is crucial. ≡≡

Thomas Rafferty, a 24-year veteran of the CAD/CAM and engineering industries, joined Auto-trol Technology in 1991 to spearhead and direct the development of Mozaic, the first object-oriented, standards-based platform for the CAD/CAM marketplace. As Vice President of Marketing and Systems development, Rafferty is currently head of Auto-trol's mechanical engineering business unit. He can be reached at Auto-trol, 125000 N. Washington, Denver CO, 80241-2400 or by phone at 303.452.4919.

DISTRIBUTED INFORMATION ≡≡

The quest for value



Tim Andrews

Distributed information is a topic of vital interest to almost all businesses today. The title of this column reflects this interest and the value brought to object technology, including object databases, by enabling distributed information. Although as the founder of a vendor I have my biases, I want to address more than the specific areas of technology related to my organization.

NEW TECHNOLOGY AND ROI

Too often those of us participating in the creation of new technologies sell the technology as the solution. We try to create successful products rather than successful solutions. We encourage our customers to use object technology to increase productivity or gain better reuse of software. These benefits, far too vague, are offered by every new technology that comes along. We must place more emphasis on understanding the business goals driving customer organizations and quantifying the benefits that object technology provides in addressing business goals: the hard return on investment (ROI).

Consider a bank that wants your money. If you were told the bank would give you back less than you gave, or merely that you would earn "a great return," you would not deposit your money. You would want to know, in advance, the ROI, e.g., 5% per year compounded daily. Now you can make relative comparisons. If another bank offered 6% for the exact same deposit, you would go there. When different investments are compared, the evaluation process is more complicated. A stock may offer a 3% dividend but greater potential for capital appreciation, perhaps 10%. The ROI could be 13%, a much better return than the 6% offered by the bank, but the risk is quite different because only a 3% dividend is assured.

During this evaluation we do not examine how the bank or the stock provides the return. If we wish to better understand the differences between the two instruments, then it is necessary to examine the "technology" underlying each security. We examine the credit worthiness of the bank: Does it carry insurance? How does it invest the money we deposit? We examine the financial statements of the company issuing the stock: What were its revenues and earnings in previous years? What business is it in? This improves our ability to choose the investment that will best satisfy our objectives.

Our attitude towards software should be the same as toward a securities investment. We are going to purchase the software, so we should think first about the ROI we expect. When comparing software packages we should evaluate the expected returns and risks associated with each package and how these factors relate to our business objectives. Once we have purchased software and begun to use it, we should measure our returns over consistent time periods so that we can continually evaluate the actual return.

Both vendors and customers need to participate in the process of evaluating and measuring ROI. In practice, however, it is very uncommon for either vendor or customer to conduct evaluations or measure ROI once a development project is under way. This is because software is generally evaluated and purchased by technology personnel whose natural inclination is to examine technological aspects and base a selection solely on those aspects. Vendors wishing to sell their products focus on marketing technology advantages to have the best chance of winning the sale.

CHANGING THE GAME

This process of technology adoption is one reason some technologies, such as AI, are failures in the marketplace. New technology has substantial risk associated with its adoption, so an ROI-based evaluation is crucial to successful adoption and eventual use of the new technology within an organization. The focus on ROI causes a change in the perspectives of both customers and vendors. Customers begin to examine business objectives, cultural effects of new technology, and support of business applications developed with the new technology. Vendors expend more effort understanding the customer's objectives and what hurdles must be overcome for the technology to make a positive contribution. A partnership results when both sides are focused on the common objective of improving the customer's business operations in a tangible way. This partnership translates directly into ROI.

As an example, let's take a brief look at distributed information systems. Many businesses today are interested in moving toward distributed systems, whether client/server systems connecting PCs to mainframes or distributed applications running in fast LANs on RISC workstations. At the highest level, the arguments are compelling:

- One can purchase MIPs for a much lower price on a PC or workstation than a host system.
- Machines can be added incrementally, giving business greater flexibility in purchasing equipment.

hotline on OBJECT-ORIENTED technology

Back issues

All back issues of the HOTLINE are available. Please call 212.274.0640 for details.

Vol. 4, No. 1/November '92 ≡ Combining object technology with data standards for the next industrial revolution ≡ Constant quality management ≡ Evolving markets for software components ≡ The quest for value ≡ Reviewing OOSE: a use case-driven approach

Vol. 3, No. 12/October '92 ≡ ROI: development environments for the lifecycle ≡ Selecting the right object-oriented method ≡ Choosing an object-oriented language ≡ Object database technology: who's using it and why? ≡ Objects and reuse

Vol. 3, No. 11/September '92 ≡ Developing strategic business systems using object technology ≡ Object training: harder than it looks ≡ Object-oriented ROI: extending the CRC across the lifecycle ≡ What TQM means for OT

Vol.3, No.10/August '92 ≡ Object technology: toward software manufacturing ≡ Return on investment: software assets and the CRC technique ≡ Object-oriented technology in Japan ≡ Providing commonality while supporting diversity

Vol.3, No.9/July '92 ≡ OOD: Research or ready ≡ Enterprise modeling: an object approach ≡ OMG's 18–24 month view ≡ Design for object-oriented applications: a CASE for wishful thinking . . .

Vol.3, No.8/June '92 ≡ Business in the Information Age ≡ From data modeling to object modeling ≡ How frameworks enable application portability ≡ Interview with Vaughan Merlyn

Vol.3, No.6/April '92 ≡ Thinking the unthinkable: reducing the risk of failure ≡ Mitigating madness with method: first establish what you value ≡ Championing object technology for career success in the 1990s ≡ Objects and actions in end-user documentation

Vol.3, No.5/March '92 ≡ TA large-scale users' assessment of object orientation ≡ Report on the Object-Oriented COBOL Task Group ≡ Interview with K.C. Branscomb

Vol.3, No.4/February '92 ≡ The big prize: acceptance of O-O by the MIS community ≡ Retrospective: 1991—the year it all changed ≡ Making the transition to O-O technology ≡ Interview with Beatriz Infante

Vol.3, No. 3/January '92 ≡ Enterprise object modeling: knowing what we know ≡ Adopting objects: pitfalls ≡ Adoption rate of object technology: a survey of NSW industry

Vol.3, No. 2/December '91 ≡ Accepting object Technology ≡ Adopting objects: a path ≡ Incorporating graphical content into multimedia presentations

Vol.3, No. 1/November '91 ≡ Leading the U.S. semiconductor manufacturing industry toward an object-oriented technology standard ≡ Coping with complexity: OOPS and the economists' critique of central planning ≡ Choosing Object Technology: What's the object? ≡ OOP: the MISsing link

Vol.2, No. 12/October '91 ≡ A modest survey of OOD approaches ≡ What is a "certified" object programmer? ≡ Perspective: investing in objects today ≡ Object oriented in Melbourne, Australia ≡ The Object Management Group

Vol.2, No. 11/September '91 ≡ From applications to frameworks ≡ Report on the Object-Oriented COBOL Task Group ≡ Getting started with object technology: efffectively planning for change ≡ Object statistics on the way ≡ On objects and bullets

Vol.2, No. 10/August '91 ≡ Distributed object management: improving worker productivity ≡ Getting the best from objects: the experience of HP ≡ APPLICATIONS: EC employs object technology ≡ CAPACITY PLANNING: Fiddling while ROMs burn

Vol.2, No. 9/July '91 ≡ Multimedia is everywhere! ≡ Developing an object technology prototype ≡ Object-oriented capacity planning ≡ How OOP has changed our developmental lifecycle ≡ Modularization of the computer system

Vol.2, No. 8/June '91 ≡ Domain of objects: the Object Request Broker ≡ Object-based approach to user documentation ≡ Report on the Object-Oriented COBOL Task Group ≡ Do we need object-oriented design metrics?

Vol.2, No.7/May '91 ≡ Hybrid object-oriented/functional decomposition for software engineering ≡ So, what makes object databases different? (Part 4) ≡ Using the generic application to solve similar domain problems ≡ Experiences using CLOS ≡ International Conference on Object-Oriented Technology, Singapore

Vol.2, No.6/Apr. '91 ≡ An artist's perspective of programming ≡ So, what makes object databases different? (Part 3) ≡ Moving from Pascal to C++, Part 3 ≡ Object projects: what can go wrong ≡ Reflections from LOOK-'91

SUBSCRIBE NOW TO THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY— DON'T MISS ANOTHER VALUE-PACKED ISSUE!

❑ *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)*

(outside US add $30 per year for air service)

Back issues @ $25 each *($27.50 foreign):*

Vol.2, Nos. _____

Vol.3, Nos. _____

❑ Phone/fax order
Call Subscriber Services at 212.274.0640
or fax this form to 212.274.0646

❑ Bill me

❑ Check enclosed
Make check payable to the HOTLINE and mail 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)

❑ Credit card orders
❑ MasterCard ❑ Visa ❑ AmEx

Card# Expiration Date

Signature

Name

Title

Company/Mail Stop

Street/Building#

City/Province

ST/Zip/Country

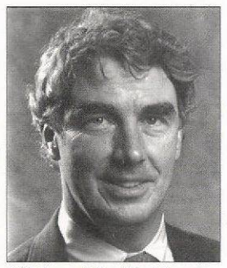
Telephone

D2KC

hotline on OBJECT-ORIENTED technology

VOL. 4, NO. 2 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES DEC. 1992

Achieving zero-cost portability today



Richard L. Dellinger

Portability is defined as the ability to design and implement a computer program on one machine and execute it on a different machine. True portability is performing this process without having to make any changes to or perform any work on the original software. Implied in the notion of true portability is that, while absolutely no work is done to move the software from one machine to another, the ported software conforms with the guidelines or rules of the host platform. We call this zero-cost porting.

Providing zero-cost porting touches many aspects of a platform. When software moves from one platform to another, we expect to fully utilize the host processor, operating system, communications capability, window management, tool kit for the user interface and multi-media resources, and access to external sources of information. Utilization of the operating system means that applications are independent of different file structures and approaches to memory management, while employing host fonts, graphics, and color.

Why is portability interesting? Primarily, true portability decreases or, better yet, completely eliminates the costs of deploying software across multiple platforms. Porting costs typically are for engineers, tools for compiling and testing, configuration management, and release control. In other words, the need to have engineering resources dedicated to each supported platform is eliminated. Portability also reduces the long-term risk of inevitable changes to the underlying platform. An informal poll of software engineering managers throughout the industry revealed that it is not uncommon to apply over 60% of an organization's development resources to performing and supporting ports to multiple platforms.

By way of example, let's look at two models for porting. In the first, you recompile your application code on each new platform and rely on similarities in compilers, libraries, and platform primitives for graphics, color, window management, and so on. You need the compiler for the new platform; you need to run the test suites; you need a debugger for subtle errors; and you need people who know about the operating environment and its tools. This is the traditional approach and is not zero cost.

In the second model for porting, you simply transfer your application executable code, unchanged, to other platforms. Each platform includes a special layer of software, usually called a virtual machine, that transforms applications operations into platform-specific actions. With no additional effort, your application executes as expected. Since exactly the same application code is executing on the target machine, testing the port is completely unnecessary. From the application software developer's point of view, all platforms are identical. Simply stated, the responsibility for porting, testing, and supporting the port has been moved from each application developer to the vendor that supplies the zero-cost portability technology.

continued on page 8

IN THIS ISSUE

FROM THE EDITOR

Last month's editorial addressed large corporate IT interests in object technology. We raised the issue of vendor responsiveness to IT's very immediate concerns about management, scaling, robustness, security, integration, and business effectiveness of object technology. We talked about the questions that business needs to have answered before effective use can be made of object technology. We discussed the frustration experienced by large corporations when vendors explain and justify object technology in terms of programming-level concepts, obviously not understanding the business impact of migration and integration issues accompanying this technology. Corporate developers—and the business decision makers who fund their work—are suffering from buzz-word–burnout. Too many technology phenomena have come and gone, leaving behind only a new (usually incomprehensible) vocabulary and incredible costs with little return.

OOPSLA was held inVancouver B.C., Canada, in October of this year. For those who have never attended, OOPSLA is ACM's annual conference on object-oriented programming systems, languages, and applications. Since its inception, OOPSLA has had a primarily academic leaning and has been well attended by technical staff and academics. In recent years, however, OOPSLA has hosted a growing vendor show floor. The presentations and papers have increasingly addressed issues of technology use as opposed to doctoral and technological research. This, combined with a growing interest in object technology on the part of the business community and corporate IT professionals, has brought corporate developers to OOPSLA in unprecedented numbers.

Ms. Elizabeth Sevean, as a partner with the Integration Consortium and founder of the Business-Technology Exchange (a user group of businesses interested in object technology), is in a good position to judge this firsthand, and served as my co-writer on this editorial.

This year's OOPSLA seemed unprepared to address the information needs of the business community. To be fair, OOPSLA is not alone in this shortcoming. Professionals from large IT organizations first attending object technology conferences have repeatedly walked away with their business and integration questions unanswered.

Vendors and most technology conferences are not addressing large business IT concerns. The industry and conference community are told that IT concerns are not being addressed because IT is not buying yet. How, we ask, can this industry expect corporate IT to buy a technology that does not address corporate IT needs?

Contrast this with the significant turnout of traditional corporate IT organizations at object technology seminars hosted exclusively for IT managers. Firms like AT&T, Prudential, General Electric, Chase Manhattan, British Airways, and American Airlines are sending employees to conferences, seminars, and training in unprecedented numbers. Organizations for IT professionals like the Data Administration Management Association, which traditionally focuses on data administration, business modeling, repositories, information architecture, etc., in the large corporate IT environment, are sponsoring seminars about object technology for their membership and experiencing turnouts well into the hundreds. Even the DB2 user groups are clamoring for speakers on object technology.

The object industry seems to expect IT to buy products before it is educated. IT has been burned by that approach too many times, and knows all too well the costs and risks of playing "do-it-yourself" integrator with new technology. IT professionals today want to be educated before they buy, as evidenced by their attendance at conferences and professional organization seminars. We suggest that the object industry could profit from recognizing this need as an opportunity. What is the road block? Why is the large IT customer being neglected?

To market a product, the vendor must assess the needs of the marketplace. It appears that the object industry has not done an accurate market assessment of IT. Interested customers are being missed. Remember that in addition to software and hardware, consulting, training, and conference registrations are all products. When prospective customers are not addressed, an opportunity is missed in proportion to the size and net profitability of the sales involved. The real measure of success is how little profitable business we are missing, not how much we are capturing.

Furthermore, marketing a product requires two way communication. The vendor must understand the customer's needs and be able to deliver a product that addresses those needs in a way the customer understands and values. We are seeing some products that reflect this realization. New development tools and concepts from IntelliCorp, Knowledge Systems Corporation, Objective: Inc., ParcPlace Systems, Persistence, and Softeam are among those forging a path for corporate IT. Among these offerings are evolving solutions to basic issues of concern to IT like reuse management, business-object analysis, automated connections to database management systems, encapsulation of networking and host graphical user interfaces, zero-cost portability, and encapsulation of legacy systems. Companies like these are making great strides in their respective areas of strength. From the viewpoint of mainframe IT, however, the object technology industry still has a long way to go.

We believe the object industry as a whole has not yet made the effort to educate itself about the needs of IT in the large-scale information-dependent enterprise. There are urgent needs in today's business that would provide outstanding opportunities for using object technology. Consider some examples: decision support, componentized process control, reconfigurable manufacturing lines, distribution of data and processing that comes with downsizing, and revitalizing legacy systems currently constrained in their usefulness by inflexible user inter-

get by for a while, but sooner or later there is going to be a bad accident. . . .

The overselling of object technology, Jan Steinman, OBJECT MAGAZINE, 9-10/92

. . . Already, object technology is reshaping programming so that it will become the integration of enterprise services. . . . Will object technology overtake conventional methods? Probably not, at least in the near future. In fact, one of the strengths of object technology is its ability to complement and coexist with conventional programs. . . .

Distributed enterprise networking: Applying object technology to distributed networking yields incrementally greater benefits, Steven Brockman, OBJECT MAGAZINE, 9-10/92

. . . Process change will come about only if we use the object-oriented approach, from requirements to coding. . . But I predict no improvement in return on investment if both methodology and management aspects are not considered concurrently. . . .

Insider: Software help wanted: revolutionary thinkers, Annie Kuntzmann-Combelles, IEEE SOFTWARE, 9/92

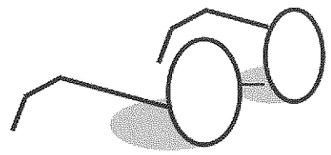
As a panelist at a recent software methods conference, I was asked, "How long will it take for the software industry to fully realize the benefits of object-oriented technologies?" My answer was and is, "Who cares, as long as you and your organization reap some benefits as soon as possible?" Implicit in my answer are a few truths. The first is I don't know how long it will take to fully realize the benefits of O-O. . . . The second truth in my answer is that the question reflects false antihype. . . designed to impede the progress of a new idea. It is usually generated by those with a vested interest in the status quo. True antihype is designed to help a new idea. It deflates unreasonable expectations and allows old ideas to be used in service to the new. Yes, there are risks to adopting object technologies. However, there are also risks with maintaining the status quo: more failed analysis eforts, more bad code, and self-inflicted increases in the project backlog. . . .

Antihype: Wait for everyone else and you'll be history, John Palmer, OBJECT MAGAZINE, 9-10/92

. . . Objects are an effective way to manage system complexity, providing data and program abstraction and a convenient way of modifying the software. However, in and of itself, the object-oriented model does not guarantee timing preditablity under both normal and abnormal operating conditions—a primary requirement of real-time software. . . By combining compiler-based static analysis and object orientation, we are giving real-time-system developers a tool that enhances timing predictability without sacrificing good engineering practices. . . .

Compiler support for object-oriented real-time software, Prabha Gopinath, Thomas Bihari, Rajiv Gupta, IEEE SOFTWARE, 9/92

BOOK WATCH



The Object Management Group

The Object Management Group announced the completion of an upgraded version of its OBJECT MANAGEMENT ARCHITECTURE (OMA) GUIDE. It contains a newly endorsed Object Model for all OMG specifications. The OMA GUIDE, available for $50, contains a reference model that is the central design guideline OMG uses for the creation of a distributed object computing environment. In addition to the new Object Model, the guide contains a complete glossary of object terms and the technical objectives of the OMG.

492 Old Connecticut Path, Framingham, MA 01701, 508.820.4300, fax: 508.820.4303

In a move that signaled the beginning of a new phase in its multiplatform cooperative computing strategy, HP stepped up ts plans to deliver what it's calling a Distributed Object-Computing Environment. This environment would permit HP users to develop applications across heterogeneous, distributed networks, without regard to operating system or hardware platform. A new HP organization, the Distributed Object Computing Program, will assume responsiblity for the development and coordination for the new distributed environment. . . .

HP defines distributed object computing environment, **HP PROFESSIONAL**, 9/92

Motorola's Microprocessor & Memory Technologies group is teaming with Distributed Systems International to provide software to manage Motorola's Fiber Distributed Data Interface (FDDI) chipset. The software, developed by Wheaton, Ill.-based DSI, is an object-oriented package that connects and disconnects stations to the FDDI ring, monitors network operations, isolates network faults, and detects conditions such as duplicate addreses that inhibit ring operation. . . .

Data Net, Andrew Collier, **ELECTRONIC NEWS**, 8/31/92

CLIENT/SERVER

. . . Today's CORBA-compliant DOM environments are a step closer to the vision of "Information at Your Fingertips": they are heterogeneous, simple to understand, and dynamic. They have a single, simple interface to heterogeneous hardware and software objects and a single, simple solution for how those objects talk to each other. . . . Hardware and software can be replaced or upgraded easily with minimal impact to users. The systems themselves can evolve over time. . . .

Client/server computing: Recent developments in OT indicate an evolution toward truly open systems, Molly Johnston, **OBJECT MAGAZINE**, 9/10/92

. . . On the client side, object-oriented companies are aiming to deliver an application package like other databases but with the focus on objects instead of the relational model. On the UNIX server side, though, there is a battle plan. When Next and other UNIX vendors embed Object Design's technology directly into their operation systems, they are hoping it will give UNIX systems more than just the abilitiy to store data in the operating system as objects rather than files. They are hoping it will also give them a leg up on Windows.

Object-oriented technology takes a front seat, **UNINEWS**, 8/24/92

VISUAL PROGRAMMING

. . . However, visual programming techniques on their own do not necessarily improve the productivity of programmers. It is through the fusion of object-oriented software development concepts and a visual development environment that both the number of programmers capable of application development and their productivity can be increased. . . .

Visual programming: Interactive construction of programs speeds developement and increases productivity, Dave Mandelkern, **OBJECT MAGAZINE**,9-10/92

VIRTUAL REALITY

. . . How would youlike to display obejct-oriented information, e.g., OOP sources, using VR and literally walk through your code? VR and objects are one and the same, just different ways of looking at data. Remember that what VR tries to do is create a possible wourld, populate it with objects, and define their behaviors and the nature of their interactions. . . .

Virtual reality: Redefining the meaning of human-computer interaction, David A. Smith, **OBJECT MAGAZINE**, 9-10/92

HAVE YOU EVER SEEN A PARADIGM SHIFT?

. . . However, the quickest way to get into trouble is to view object technology as a panacea by taking any of its advantages for granted. Assuming specific benefits, especially in the early stages, causes problems if those benefits are not actively pursued throughout the project. The proper level of "buy in" is crucial—not enough, and a critical mass is not achieved. On the other extreme, there is a difference between "buying in" and jumping off a cliff! Perhaps most importantly, changing technology without changing culture is like switching to automobiles while retaining horse-and-buggy protocol—you can

faces and data structures. Until the object industry understands and communicates with the IT customer, sales will not be forthcoming.

Comments from experienced IT professionals may be summed up best by analogy: Object technology has presented IT with a wonderful "erector set" full of components, yet completely lacking in assembly and integration instructions for these components. We would suggest that IT is not buying simply because no one is providing the instructions for assembling the pieces in the context of their existing environment, concerns, and needs.

Current object methodologies do not provide adequate instructions for dealing with the business. They do not identify that something (the business) exists above and beyond the boundaries of the applications we are building with OOPLs. Most of IT's current problems exist precisely because of this application tunnel vision. Instead of focusing on the business, we IT professionals have been taught to think about computer solutions. Our traditional development lifecycles do not revalidate the application against the business at the enterprise level. The topic of business reengineering at the enterprise level recently has become an extremely hot topic in the IT environment. To make the best use of object technology in the IT environment, we must avoid application tunnel vision. Our approach to objects in the IT environment must effectively address business object identification, sharing, management, etc. With reuse and flexibility as key selling points of object technology, revalidation to the business is critical.

How will the 80% of the potential marketplace represented by IT get the object industry's attention to real business issues? What will motivate development of a structure for the implementation and integration of object technology? To date, no one has put together a structure to satisfy the needs of IT and large organizations with mainframe and distributed environments. 60% of mainframe IT organizations will undertake an OT project in 1993, as opposed to 40% of IT organizations overall. It's the IBM mainframe shops that are hurting badly enough to put in the effort—the pain is forcing them to change. These are also the people migrating off their mainframes, variously terming their efforts downsizing, cooperative computing, client-server computing, etc. The market segment with the largest problem to solve is actively looking for and experimenting with solutions.

As for those vendors and conference promoters who believe there is no interest in the mainframe IT environment, we have been asking questions and raising issues at every object technology conference this year, as well as at several vendor-delivered training seminars. Mainframe IT professionals keep coming up and telling us about the same concerns, which are not being addressed. We think vendors do not see the market because they are largely unfamiliar with the problems of the mainframe environment; they have not packaged their products in a way that addresses IT's concerns. Until vendors recognize and correct this, mainframe IT organizations will see no value in buying their products. Some research in this area would be appropriate and profitable.

Speaking of profitable, in this month's issue Mr. Richard Dellinger of ParcPlace Systems addresses the financial savings from zero-cost portability. The concept is simple, really. Instead of writing every user program and software package in a raw third-generation language (3GL) like COBOL, C, or C++, developers construct their products in a development environment that provides a virtual machine. The virtual machine is an abstract computer application program interface (API) that, through significant effort on the part of vendors like ParcPlace Systems, make all supported computers and their operating systems look identical from the application program's point of view.

Stable, well-established virtual machine development environments are available from several vendors. All are in active use for development of commercial products and IT applications. Smalltalk virtual machines are available from ParcPlace and Digitalk. Objective: Inc. provides a fifth-generation language development environment product called Macroscope. Each of these products is available on some combination of the following host systems: MS/DOS, MS/Windows, Macintosh, NEXT, SUN, OS/2, RS/6000. Read Mr. Dellinger's article carefully. The cost-effectiveness of his proposal cannot be overstated for the corporate developer building client-server applications or the

continued on page 9

hotline on OBJECT-ORIENTED *technology*

Robert Shelton, Editor

SIGS ADVISORY BOARD
Tom Atwood, Object Design
Grady Booch, Rational
George Bosworth, Digitalk
Brad Cox, George Mason University
Chuck Duff, Symantec
Adele Goldberg, ParcPlace Systems
R. Jordan Kreindler, General Electric
Meilir Page-Jones, Wayland Systems
Tom Love, OrgWare, Inc.
Bertrand Meyer, Interactive Software Engineering
Sesha Pratap, CenterLine Software
P. Michael Seashols, Versant Object Technology
Bjarne Stroustrup, AT&T Bell Labs
Dave Thomas, Object Technology International

HOTLINE EDITORIAL BOARD
Jim Anderson, Digitalk, Inc.
K.C. Branscomb, Lotus Development Corp
Mary E.S. Loomis, Versant Object Technology
Reed Phillips, Knowledge Systems, Corp.
Bernadette G. Reiter, Objective: Inc.
Steven Weiss, Wayland Systems
John A. Zachman, Zachman International

SIGS Publications, Inc.
Richard P. Friedman, Founder & Group Publisher

ART/PRODUCTION
Kristina Joukhadar, Managing Editor
Susan Culligan, Pilgrim Road, Ltd., Creative Direction
Elizabeth A. Upp, Production Editor
Jennifer Englander, Art/Production Coordinator

CIRCULATION
Ken Mercado, Fulfillment Manager
Vicki Monck, Circulation Assistant
John Schreiber, Circulation Assistant

MARKETING
Amy Friedman, Projects Manager
Lorna Lyle, Promotions Manager—Conferences
Sarah Hamilton, Promotions Manager—Publications
Caren Polner, Promotions Graphic Artist

Administration
David Chatterpaul, Bookkeeper
Ossama Tomoum, Business Manager
Margherita R. Monck, General Manager

Jane M. Grau, Contributing Editor

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012. (212)274-0640. Mailed First Class. Subscription rate — one year (12 issues) $249, Foreign and Canada $279. Single copy $25.

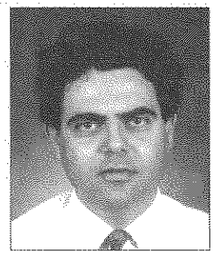
POSTMASTER: Send address changes & subscription orders to HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville, NJ 07834.

Submit editorial correspondence to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123 voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of HOTLINE ON OBJECT-ORIENTED TECHNOLOGY, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, THE X JOURNAL, C++ REPORT, THE SMALLTALK REPORT, and THE INTERNATIONAL OOP DIRECTORY.

Design by contract: building bug-free O-O software



Bertrand Meyer

When evaluating a software development method, many people tend to view productivity as the major expected benefit. For object-oriented technology, I believe this is inappropriate. Who really cares about the number of lines programmers churn out each month? What matters is how good these lines are. In other words, the focus should be less on productivity than on quality. Better software methods, languages, and tools should also better productivity, of course, but mainly as a by-product of improved quality. In the words of K. Fujino, Vice President of NEC Corporation's C&C Software Development Group, "When quality is pursued, productivity follows."[1]

A major component of quality in software is reliability: a system's ability to perform its job according to the specification (correctness) and to handle abnormal situations (robustness). Put more simply, reliability is the absence of bugs.

Reliability, although desirable in software construction regardless of the approach, is particularly important in the object-oriented method because of the special role given by the method to reusability: unless we can obtain reusable software components whose correctness is much more trustworthy than that of usual run-of-the-mill software, reusability is a losing proposition.

How can we build reliable object-oriented software? The answer has several components. Static typing, for example, is a major help for catching inconsistencies before they develop into bugs. By itself, reusability also helps: if you are able to reuse component libraries produced and (presumably) validated by a reputable outside source, rather than developing your own solution for every single problem you encounter, you can start trusting the software as much as the machine on which it runs. In effect, the reusable libraries become part of the "hardware-software machine" (hardware, operating system, compiler).

But this is not enough. To be sure that our object-oriented software will perform properly, we need a systematic approach to specifying and implementing object-oriented software elements and their relations in a software system. This article introduces such a method, known as Design by Contract. Under the Design by Contract theory, a software system is viewed as a set of communicating components whose interaction is based on precisely defined specifications of mutual obligations, or contracts.

The benefits of Design by Contract include the following:

- a better understanding of the object-oriented method and, more generally, of software construction
- a systematic approach to building bug-free object-oriented systems
- an effective framework for debugging, testing and, more generally, quality assurance
- a method for documenting software components
- a better understanding and control of the inheritance mechanism
- a technique for dealing with abnormal cases leading to a safe and effective language construct for exception handling

The ideas developed below are part of Eiffel,[2,3] which the reader is urged to view here not so much as a programming language but rather as a software development method. A longer exposition of the approach may be found in a recent article.[7]

SPECIFICATION AND DEBUGGING

As a key step toward improving software reliability, it is important to realize that the first and perhaps most difficult problem is to define, as precisely as possible, what each software element is supposed to do. Of course, specifying a module's purpose will not ensure that it will achieve that specification, but, conversely, if we don't state what a module should do, there is little likelihood that it will do it (Dijkstra's law of excluded miracles).

As will be seen below, a specification, even if it does not fully guarantee the module's correctness, is a good basis for systematic testing and debugging.

The Design by Contract theory, then, suggests associating a specification with every software element. These specifications (or contracts) govern the interaction of the element with the rest of the world.

This presentation will not, however, advocate the use of full formal specifications. Although the work on formal specifications in general[4] and as applied to the object-oriented method[5] is attractive, we will settle for an approach in which specifications are not necessarily exhaustive. This has the advantage that the specification language is embedded in the design and programming language (in this case, Eiffel), whereas formal specification languages are typically non-executable or, if they are executable, can be used only for prototypes. Here our criteria are more demanding: we want our language to be used for practical commercial development and hence yield efficient implementation. This preserves a key property of a well-understood object-oriented process: seamlessness,[6] which makes it possible to use a single notation and a single set of concepts throughout the software lifecycle, from analysis to implementation and maintenance, en-

FYI

Excerpts from leading industry publications on aspects of object technology

. . . [Intek Integration Technologies, Inc. of Bellevue, Washington] had only its three chief architects learn the object-oriented design (OOD) methodology behind OOP and C++. And it did take them six months. Then those three created the class libraries needed for Intek's software. The other nine programmers in the company just learned C++ syntax and were taught how to string together the class libaries. It took them about two weeks to master that. Intek now uses C++ to do all its factory automation projects. . . .

Borland's bridge to OOP, Lee Thé, DATAMATION, 8/15/92

STANDARDS

. . . We may be looking forward to the electronic equivalent of the Tower of Babel if everyone insists on doing things their own way—trying to lock up all of the market with mutually exclusive approaches. I have a colleague who says that the need for standards is a middle-age disease. Standards are unquestionalbly dull, but they are precisely what make telephones and fax machines so useful (and widely used). We need to apply some of the same logic to the next round of operating environments. and, if objects are to have any chance of succeeding in the short term, we need to carefully think about how future software products will be developed, priced, packaged and distributed.

Industry watch:What do Microsoft, IBM and Apple have in common?, Richard Dalton, WINDOWS, 8/92

. . . Since the classes in OO software systems typically describe particular objects rather than abstract concepts, and since the field is labeled OBJECT-oriented programming, it would be a good idea to avoid the term "CLASS" in this field and use e.g., "OBJECT type" instead, because "CLASS" has too strong a connection with the area of taxonomy of concepts. . . .

Objectivism: "CLASS" considered harmful, Jürgen F.H. Winkle, COMMUNICATIONS OF THE ACM, 8/92

THE BUSINESS OF OBJECTS

. . . [Taligent's Guglielmi] wants to court UNIX users by writing "adapters" within the Pink system. IBM and Apple will write adapters so the operating system will be portable, making existing software able to run on it. But analysts predict that even if Pink is that advanced, any initial success will be limtied to IBM and Apple machines. To convince system vendors and software developers to write for Pink, Guglielmi can point to a large installed base of machines, something that another bleeding-edge company couldn't. . . .

From blue to pink, Gary Andrew Poole, UNIX WORLD, 10/92

. . . The software industry itself has a long way to go before it converts its thinking about packaging and pricing to an object orientation. Word for Windows is a good example: Each time it is updated it becomes a more complex package of what might, in the future, be sold as individual objects. . . . The problem is that almost no one is creating, packaging or pricing software by the object. This means an enormous upheaval somewhere along the line, not only by the companies that create the software, but throughout the entire software distribution chain, right down to the way that the neighborhood software store carves up shelf space, does inventory and Calculates its profit margin. . . .

Industry watch:What do Microsoft, IBM and Apple have in common? Richard Dalton, WINDOWS, 8/92

If a recent presentation by Microsoft Corp. executives to developers is any indication, the company is making good progress on its object-oriented environment for Windows NT, code-named Cairo . . . which will begin shipping next year. . . . [Program chief Jim Allchin]said the developers didn't like Microsoft's original user interface design for Cairo, so it will be changed to look more like Windows. And Microsoft is also reviewing its original decision to make Cairo available only on NT. A scaled-down version of Cairo, minus features such as security, may show up on DOS Windows after all.

Windows Goes to Cairo, DATAMATION, 8/15/92

Industry Briefs

K.C. Branscomb, most recently chief executive officer of **IntelliCorp, Inc.**, has been named Senior Vice President of business development at **Lotus Development Corp.** Responsible for pursuing business alliances and technology relationships in support of Lotus' workgroup computing strategy, Branscomb will report to Lotus President and CEO Jim Manzi, joining the company's senior executive staff.

AICorp Inc. and **Aion Corp.** announced the completion of their merger and will begin doing business as of Oct. 1, 1992 under the name of **Trinzic Corp.** Trinzic's management team is a combination of the leadership of both companies. The executive management for Trinzic is as follows: Robert Goldman, Chairman; Jim Gagnard, Chief Executive Officer; Frank Chisholm, President; Larry Cohn, Executive Vice President of Technical Operations; Irv Lichtenwald, Chief Financial Officer; Larry Harris, Chief Technology Officer; and Colin Phillips, Vice President of International Operations. Trinzic will continue to enhance and market its two application development tools: KBMS and The Aion Development System (AionDS).

Denny K. Paul, formerly a key senior executive at both Businessland Inc. and Dataquest Inc., has joined **Infinity International Financial Technology Inc.** as Chief Financial Officer. Mr. Paul joins Infinity as Vice President and CFO with responsibilities in general management, administration, and finance.

SGN and **EURIWARE** announced the formation of **INTELLITIC INTERNATIONAL**, with a charter to promote, sell, develop, distribute, and maintain the MATISSE Database Management Product, a second-generation, industrial-strength, open-semantic object database. INTELLITIC INTERNATIONAL is managed by Executive Chairman Bruno de Saint Chamas, Executive Vice President of Marketing and Sales Olivier Loubiere, and Executive Vice President of Technology Pierre Moller.

Sherpa Corporation joined the **Object Management Group (OMG).** Sherpa Corporation markets PIM solutions to Fortune 100 companies in the aerospace, automotive, defense, telecommunications, and consumer electronics industries.

SunPro and **MetaWare Incorporated** have announced a licensing agreement in which MetaWare's x86 code-generation technology is used in a new family of SunPro compilers for personal computers. The family, called ProCompiler, makes available compilers for developing applications on the Solaris for X86 operating environment.

Non-profit International Consortium for Eiffel (NICE) announced that Robert "Rock" Howard will serve as the new chairperson. Mr. Howard is the president of Rock Solid Software, which distributes Eiffel products and publishes the Eiffel OUTLOOK journal.

Object Technology International Inc. announced two new ENVY/*Developer* distributors. **Cyberdyne Systems Corporation Pty. Limited** has agreed to be an ENVY/*Developer* distributor for the Australian market, and **Artificial Intelligence International Ltd. (AIIL)** has become a distributor of ENVY/*Developer* for Objectworks/Smalltalk in the United Kingdom. Both companies have experience providing consulting and product solutions for their Smalltalk customers.

Objectivity selected Hewlett-Packard's SoftBench framework as its standard solution for integrating multiple software applications into a single cohesive environment. Under the terms of the agreement, Objectivity will incorporate the SoftBench Broadcast Message Server technology into Objectivity tools and also may offer this technology to Objectivity customers.

Data General Corporation and **NeXT Computer Inc.** announced a relationship in which Data General will resell NeXT workstations with AViiON servers. In addition, the two companies will cooperate in the further development of advanced client/server solutions. Data General will port NeXT's NetInfo to its AViiON servers by the end of 1992.

NeXT Computer Inc. and **Auspex Systems Inc.** will comarket Auspex NFS network servers to NeXT workstation customers under a teaming agreement just completed by the companies. Auspex also announced that the company will work with Xedoc Software Development, Pty. Ltd. to sell and support NeXT network management software, called NetInfo SPARC Server Edition, used with Auspex servers. NeXT and Auspex will recommend each other's products when appropriate to meet customer needs.

suring better mapping from solution to problem and thus, among other benefits, smoother evolution.

THE NOTION OF CONTRACT

In human affairs, contracts are written between two parties when one of them (the supplier) performs some task for the other (the client). Each party expects some benefits from the contract and accepts some obligations in return. Usually, an obligation for one of the parties is a benefit for the other. The aim of the contract document is to spell out these benefits and obligations.

A tabular form such as in Table 1 (illustrating a contract between an airline and a customer) is often convenient for expressing the terms of such a contract.

Table 1.

	Obligations	Benefits
Client	Be at the Santa Barbara airport at least 5 minutes before scheduled departure time. Bring only acceptable baggage. Pay ticket price.	Reach Chicago.
Supplier	Bring customer to Chicago.	No need to carry passenger who is late, has unacceptable baggage, or has not paid ticket price.

A contract document protects clients by specifying how much should be done, and suppliers by stating their non-liability for failing to carry out tasks outside the specified scope.

The same ideas apply to software. Consider a software element E. To achieve its purpose (fulfill its own contract), E uses a certain strategy, which involves a number of subtasks, $t^1, \ldots, t^n$. If subtask t^i is non-trivial, it will be achieved by calling a certain routine R. In other words, E contracts out the subtask to R. Such a situation should be governed by a well-defined roster of obligations and benefits: a contract.

Assume for example that t^i is the task of inserting a certain element into a dictionary (a table where each element is identified by a certain character string used as key) of bounded capacity. The contract will be as shown in Table 2.

This contract governs the relations between the routine and any potential caller. It contains the most important information that can be given about the routine: what each party in the contract must guarantee for a correct call, and what each party is entitled to in return.

Table 2.

	Obligations	Benefits
Client	Make sure table is not full and key is a non-empty string	Get updated table where the given element now appears, associated with the given key.
Supplier	Record given element in table, associated with given key.	No need to do anything if table is full, or key is empty string.

So important is this information that we cannot remain satisfied with an informal specification of the contract as above. In the spirit of seamlessness (encouraging us to include all relevant information, at all levels, in a single software text), we should equip the routine text with a listing of appropriate conditions. Assuming the routine is called *put*, it will look as follows in Eiffel syntax, as part of a generic class *DICTIONARY [ELEMENT]*:

```
put (x: ELEMENT; key: STRING) is
        -- Insert x so that it will be retrievable through key.
    require
        count <= capacity;
        not key.empty
    do
        ... Some insertion algorithm ...
    ensure
        has (x);
        item (key) = x;
        count = old count + 1
    end
```

The **require** clause introduces an input condition, or precondition; the **ensure** clause introduces an output condition, or postcondition. Both of these conditions are examples of assertions or logical conditions (contract clauses) associated with software elements. In the precondition, *count* is the current number of elements and *capacity* is the maximum number; in the postcondition, *has* is the boolean query that tells whether a certain element is present, and *item* returns the element associated with a certain key. The notation **old** *count* refers to the value of *count* on entry to the routine.

CONTRACTS IN ANALYSIS

The above example is extracted from a routine describing an implementation (although the notion of dictionary is meaningful independent of any implementation concern). But the concepts are just as interesting at the analysis level. Imagine, for example, a model of a chemical plant, with classes such as *TANK*, *PIPE*, *VALVE*, *CONTROL_ROOM*. Each of these classes describes a certain data abstraction—a certain type of real-world object, characterized by the applicable features (operations). For example, *TANK* may have the following features:

- Yes/no queries: *is_empty*, *is_full*...
- Other queries: *in_valve*, *out_valve* (both of type *VALVE*), *gauge_reading*, *capacity*...
- Commands: *fill*, *empty*,...

To characterize a command such as *fill*, we may use a precondition and postcondition as above:

```
fill is
        -- Fill tank with liquid
    require
        in_valve.open;
        out_valve.closed
    deferred -- i.e., no implementation
    ensure
        in_valve.closed;
        out_valve.closed;
        is_full
    end
```

This style of analysis avoids a classic dilemma of analysis and specification: either you use a programming notation and run the risk of making premature implementation commitments, or you stick with a higher-level notation ("bubbles and arrows") and remain vague, forsaking one of the major benefits of the analysis process: the ability to state and clarify delicate properties of the system. Here the notation is precise (thanks to the assertion mechanism, which may be used to capture the semantics of various operations) but avoids any implementation commitment. (There is no danger of such a commitment in the above example, since what it describes includes no software and, indeed, no computer yet! Here we are using the notation just as a modeling tool.)

Jean-Marc Nerson's BON object-oriented analysis and design method[7,8] starts from these ideas but provides complementary representations (graphical, tabular in the style of Wirfs-Brock et al.[9] and a multi-step methodological sequence. An unpublished work by Mark Ratjens from Class Technology (Sydney, Australia) is based on similar premises.

INVARIANTS

Preconditions and postconditions apply to individual routines. It is also important to use assertions to characterize a class as a whole, rather than its individual routines. An assertion describing a property that holds all instances of a class is called a class invariant. For example, the invariant of *DICTIONARY* could state:

```
invariant
  0 <= count; count <= capacity
```

and the invariant of *TANK* could state that *is_full* really means "is approximately full":

```
invariant
  is_full = (0.97 * capacity <= gauge)
    and (gauge <= 1.03 * capacity) ...
```

Class invariants are consistency constraints characterizing the semantics of a class. This notion is important as a basis for configuration management and regression testing because it describes the deeper properties of a class: not just its characteristics at a certain moment of evolution, but the constraints that also must apply to subsequent changes. In my view, the notion of class invariant is one of the three or four most important concepts in the whole object-oriented approach.

Viewed from the contract theory, an invariant is a general clause that applies to the entire set of contracts defining a class.

DOCUMENTATION

An important application of contracts is that they provide a standard way to document software elements: classes. To provide client programmers with a proper description of the interface properties of a class, it suffices to give them a version of the class, known as the short form, which is stripped of all implementation information but retains the essential usage information: the contract.

The "short" form (provided as one of the formats for displaying the class text by the browsing tools of the environment) retains headers and assertions of exported features, as well as invariants, but discards everything else. For example:

```
class interface DICTIONARY [ELEMENT] feature:
  put (x: ELEMENT; key: STRING) is
    -- Insert x so that it will be retrievable through key.
  require
    count <= capacity;
    not key.empty
  ensure
    has (x);
    item (key) = x;
    count = old count + 1
  ... Interface specifications of other features
invariant
  0 <= count; count <= capacity
end class interface -- DICTIONARY
```

This short form serves as the basic tool for documenting libraries and other software elements. It also serves as a central communication tool between developers. We have found that emphasis on the short form facilitates software design and project management, encouraging developers and managers to discuss key issues (interface, specification, intermodule protocols) rather than internal details.

TESTING, DEBUGGING, AND QUALITY ASSURANCE

Given a class text equipped with assertions, we should ideally be able to prove mathematically that the routine implementations are consistent with the assertions. In the absence of realistic tools to do this, we can settle for the next best thing, which is to use assertions for testing.

Compilation options enable the developers to determine, class by class, what effect assertions should have, if any: no assertion checking (under which assertions have no effect at all, serving as a form of standardized comments), preconditions only (the default), preconditions and postconditions, all of the above plus class invariants, or all assertions.

These mechanisms provide a powerful tool for finding mistakes. Assertion monitoring is a way to check what the software does against what its author thinks it does. This yields a productive approach to debugging, testing and quality assurance, in which the search for errors is not blind but based on consistency conditions provided by the developers themselves.

CONTRACTS AND INHERITANCE

An important consequence of the contract theory is a better understanding of the central object-oriented notions of inheritance, polymorphism, redefinition and dynamic binding.

A class *B* that inherits from a class *A* may provide a new declaration for a certain inherited feature r of A (Fig. 1). For example, a specialized implementation of *DICTIONARY* might redefine the algorithm for *put*. Such redefinitions are potentially dangerous, however, as the redefined version could in principle have a completely different semantics. This is particularly worrisome in the presence of polymorphism, which means that in the call *a.r* the target *a* of the call, although declared statically of type *A*, could in fact be attached at runtime to an object of type *B*. Then dynamic binding implies that the *B* version of *r* will be called in such a case.

This is a form of subcontracting: *A* subcontracts r to *B* for tar-

Sun Pro

SunPro announced the availability of the ProWorks family of development environments for the Solaris for X86 operating environment, the initial offerings providing the principal development component of the Solaris for X86 early access kit. The ProWorks family of integrated development environments for C, C++, and Fortran consists of the ProWorks tools and a ProCompiler language system for C, C++, or Fortran. The ProWorks tools, which use the OPEN LOOK graphical user interface, include Session Manager, Debugger, SourceBrowser, Analyzer, Make Tool, and FileMerge. Early versions of ProWorks development environments including the ProCompiler language systems are available immediately through SunSoft's Solaris for X86 early access kits.

SunPro, 2550 Garcia Avenue, Mountain View, CA, 94043-1100, 415.960.1300, fax: 415.969.131

Intellitic International

Intellitic International is pleased to announce the release of MATISSE Version 2.1, a second-generation Open Semantic Object Database. This multiuser client/server architecture is compatible with C++ and fully complies with the OODBMS Manifesto. Its open architecture is also in compliance with ANSI standards. With MATISSE, the developer is able to use any third-party development tool and any common language, such as C, C++, FORTRAN, ADA, and COBOL. Built-in features include cardinality constraints, type checking, triggers, and daimons. Based on the ANSI three level architecture, modularity is inherent within any MATISSE application and referential integrity is guaranteed by the MATISSE Meta-Schema.

Intellitic International, USA: ODB (Object Databases),
238 Broadway, Cambridge, MA 02139, 617.354.4220, fax: 617.547.5420
Corporate Office Intellitic International: Saint Quentin en Yvelines, Cedex, France, +33.1.30.14.54.30

Lucid Inc.

Lucid Inc. announced the first of several scheduled enhancements to the Energize Programming System. Energize 1.1, with support for more tools and utilities and improved performance in its native code C++ and C compiler, is shipping immediately. New releases of Lucid C and Lucid C++ and compilers are also shipping immediately. Pricing is unchanged from Lucid's original offering that allows each programmer in a workgroup to have his or her own product. For five-person workgroups, it is priced at $3,250 per seat. The product can also be purchased in single quantities for $4,250. Additional discounts are available for larger workgroups and sites. Current customers with support agreements receive free upgrades.

Lucid Inc., 707 Laurel Street, Menlo Park, CA 945025, 415.329.8400, fax: 415.329.8480

Microtec Research Inc.

Microtec Research Inc. announced availability of its Intel i960 and 8086-family microprocessor development tools on the IBM RISC System/6000) workstation. These tools join the Microtec Research Motorola 6800 family tools already on the IBM workstation and were ported under an agreement with IBM. A C++ cross compiler is available for 68000 target processors, compliant with version 2.1 of the AT&T specification. US list price starts at $4,300 for the MCP68K Package, which includes ANSI C Compiler, Assembler, Linker, and Librarian for Motorola 6800 processors.

Microtec Research Inc., 2350 Mission College Blvd., Santa Clara, CA 95054, 408.980.1300, fax: 408.982.8266

Sapiens Software Corporation

Sapiens Software Corporation released the 3.3 version of its Star Sapphire Common LISP for the PC, PS/2s, and compatibles. The product includes an interpreter, incremental and fastload compiler for Common LISP, EMACS editor, and over 1 Mb of online reference materials for the language. Star Sapphire support for the Common LISP Object System (CLOS) has been upgraded, with version 3.3 including most of the proposed ANSI CLOS standard. The product sells for $99.95 and has an academic volume discount program.

Sapiens Software Corporation, PO Box 3365, Santa Cruz, CA 95063, 408.458.1990, fax: 408.425.0905

TGS Systems, Ltd

TGS Systems, Ltd. announced the Prograph IAC Goodies disk, which gives developers using Prograph's object-oriented language high-level facilities for writing applications making use of Apple Events. The IAC Goodies disk is a $49 disk with documentation, classes, and examples for writing applications in Prograph that make use of the Inter Application Communications (IAC) capabilities of Macintosh System 7. Higher level support, in the form of Prograph classes, is provided for the Apple event Object Model.

TGS Systems Ltd., 2745 Dutch Village Road, Suite 200, Halifax, Nova Scotia, Canada B3L 4G7,
902.455.4446, fax: 902.455.2246

sharing of information across documents and applications; enhanced graphics; global computing; and general usability features. Added to the application development environment of Release 3.0 are the Database Kit, 3D Graphics Kit, PhoneKit, and Indexing Kit, as well as bundled Novell Client and AppleShare Client software, additional Macintosh and DOS file support, and expanded ability to share files and printers from various vendors, and to fax modems and CD-ROM drives among networked users. In addition to built-in ISDN capabilities, Hayes Microcomputer Products Inc. provides an ISDN Extender, a telecommunication network interface module that can be used for point-to-point communications and remote LAN access. Registered owners of NeXTSTEP Release 2 can upgrade to Release 3.0 for $295. NeXTSTEP Release 3.0 will be included automatically with all NeXT computers, and will also be provided on a CD-ROM for backup.

NeXT Computer Inc., 900 Chesapeake Drive, Redwood City, CA 94063 415.366.0900, fax: 415.780.3714

Micro Data Base Systems Micro Data Base Systems (mdbs) is now shipping the Object/1 Professional Pack for the Oracle relational database management system. Developed jointly by mdbs and Database Engineering Ltd., the Object/1 Professional Pack for Oracle allows developers to manage an Oracle session within an Object/1 application. Object/1 is an object-oriented development environment that allows rapid application development of graphical user interfaces (GUIs) in Windows and Presentation Manager. Object/1 applications can query and update data from Oracle through the use of Oracle's SQL language. The Object/1 Professional Pack for Oracle RDBMS is $495.

Micro Data Base Systems, Two Executive Drive, PO Box 6089, Lafayette, IN 47903-6089, 317.447.1122, toll-free: 800.445.MDBS, fax: 317.448.6428

Liant Software Liant Software announced that its PHIGS+ -based programming library—*FIGARO+ 3.0 C*—enables programmers to develop graphics applications in the object-oriented environment of C++. FIGARO+ is an independent PHIGS+ implementation, providing tight integration with the X Window System, PHIGS' Extension to X (PEX), as well as graphics accelerators like Sun Microsystems' XGL, Silicon Graphics' GL and GLX and Hewlett-Packard's STARBASE. US list prices begin at $2,600, depending on the configuration. FIGARO+ 3.0 C platforms range from PC to supercomputer. Liant also announced a major upgrade of its C-scape User Interface Management System, an object-oriented C development tool for rapidly creating portable text and graphics-based user interface (UI) applications. Developers can now create applications using the new CUA (common user access)-style borders for both text and graphics mode, as well as other advanced windowing functions. C-scape DOS now also supports Microsoft C. C++ 7l0 and Watcom C 9.0, as well as Borland C.C++ 3.X and Zortech C++ 3.0 compilers. In addition to other new features, new classes have been added and there is expanded documentation. C-scape 4.0 also includes the Oakland Graphics Library, a device-independent graphics library. North American prices start at $499, depending on configuration. Upgrade prices start at $299.

Liant Software, Framingham, MA 619.457.5359, toll free: 800.662.9866

Digitalk Inc. Digitalk Inc.'s language-neutral PARTS workbench consists of a catalog of prebuilt components, both visual and nonvisual, and a workbench window. Applications are created by first dragging parts from the catalog into the workbench, then "wiring" parts together by drawing lines between them. PARTS Workbench for OS/2 2.0 has a suggested retail price of $1,995. Current Smalltalk/V customers can contact Digitalk for a special offer. Digitalk Inc. is also shipping the 32-bit version of its object-oriented Smalltalk/V development environment for OS/2 2.0. The new version results in Smalltalk/V applications that are up to 100 percent faster and 50 percent smaller than 16-bit OS/2 applications. Smalltalk/V for OS/2 version 2.0 has other improvements, including the ability to call both 16-bit and 32-bit Dynamic Link Libraries, a debugger with enhanced single-stepping capability, improved support for bitmaps, double-byte character set characters in Smalltalk/V code, and support for OS/2's common dialog boxes. Smalltalk/V version 2.0 for OS/2 has a suggested retail price of $99.50. Individuals and small groups interested in learning object-oriented design and Smalltalk/V can now attend public training classes offered monthly by Digitalk Inc. at Digitalk Professional Services' new facility in the Portland, Oregon, area. Digitalk continues to offer a wide range of object-oriented courses and seminars for delivery at a customer's facility.

Digitalk Inc., 9841 Airport Boulevard, Los Angeles, CA 90045, 310.645.1082, fax: 310.645.1306

gets of the corresponding type. But a subcontractor must be bound by the original contract. A client executing a call under the form:

```
if a.pre then
  a.r
end
```

must be guaranteed the contractually promised result: the call will be correctly executed since the precondition is satisfied (pre is assumed to be the precondition of *r*); and on exit *a.post* will be true, where *post* is the postcondition of *r*.

The fundamental principle of subcontracting follows from these observations: a redefined version of *r* may keep or weaken the precondition; and it may keep or strengthen the postcondition. Strengthening the precondition, or weakening the postcondition, would be a case of "dishonest subcontracting" and could lead to disaster. The Eiffel language rules for assertion redefinition[6] support the principle of subcontracting.

These observations shed light on the true significance of inheritance—not just a reuse, subtyping, and classification mechanism, but a way to ensure compatible semantics by other means. They also provide useful guidance as to how to use inheritance properly.

EXCEPTION HANDLING

Among the many other applications of the contract theory, we may note that the theory leads naturally to a systematic approach to the thorny problem of exception handling, or handling abnormal cases.

A software element is always a way to fulfill a certain contract, explicit or not. An exception is the element's inability to fulfill its contract for any reason: a hardware failure has occurred, a called routine has failed, or a software bug makes it impossible to satisfy the contract.

In such cases only three responses make sense:

1. **Resumption.** An alternative strategy is available. The routine will restore the invariant and and make another attempt using the new strategy.
2. **Organized panic.** No such alternative is available. Restore the invariant, terminate, and report failure to the caller by triggering a new exception. (The caller will itself have to choose between the same three responses.)
3. **False alarm.** It is in fact possible to continue, perhaps after taking some corrective measures. This case seldom occurs (regrettably, since it is the easier to implement!).

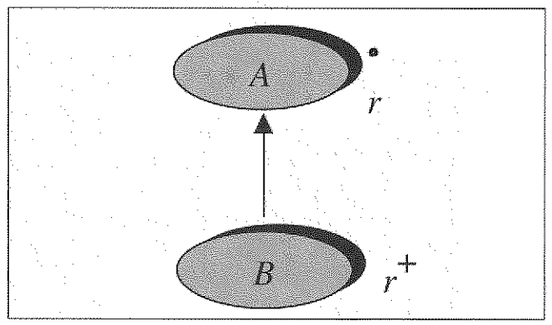


Figure 1.

The exception mechanism follows directly from this analysis. It is based on the notion of "rescue clause" associated with a routine, and of "retry instruction," which implements resumption. Details may be found elsewhere.[2,3,10] An example without further comment will illustrate the mechanism: the alternate behavior of the routine (similar to clauses that occur in human contracts to allow for exceptional, unplanned circumstances). If there is a rescue clause, any exception occurring during the routine's execution will interrupt the execution of the body (the "do clause") and start execution of the rescue clause. The clause contains one or more instructions:

```
attempt_transmission (message: STRING) is
        -- Attempt to transmit message over a communication line
        -- using the low-level (C) procedure unsafe_transmit, which
        -- may fail, triggering an exception.

        -- After 100 unsuccessful attempts, give up (triggering
        -- an exception in the caller).
    local
        failures: INTEGER
    do
        unsafe_transmit (message)
    rescue
        failures := failures + 1;
        if failures 100 then
            retry
        end
    end
```

ASSESSMENT

The theory as sketched above and explained in more detail in some of the publications listed below leaves a number of questions open. Two of the most important are:

- How do these ideas transpose to the world of concurrent object-oriented programming?
- Exactly how powerful should the assertion language be?
- How does the method of design by contract apply to the earliest stages of software investigation (domain analysis, business model, overall plan)?

But one does not need to wait for these questions to be solved. Design by Contract already has been widely applied. The theory provides a powerful thread throughout the object-oriented method and addresses, at least in part, of many of the issues that people encounter as they start applying object-oriented techniques and languages seriously: what kind of "methodology" to apply, on what concepts to base the analysis step, how to specify components, how to document object-oriented software, how to guide the testing process and, most importantly, how to build software so that bugs do not show up in the first place. Reliability should be built-in, not an afterthought. ≡≡

References

1. Ghezzi, C., M. Jazayeri, and D. Mandrioli. FUNDAMENTALS OF SOFTWARE ENGINEERING, Prentice Hall, Englewood Cliffs, NJ, 1991.

2. Meyer, B. OBJECT-ORIENTED SOFTWARE CONSTRUCTION, Prentice Hall, 1988.
3. Meyer, B. EIFFEL: THE LANGUAGE, Prentice Hall, Englewood Cliffs, NJ, 1991.
4. Hayes I.J. (Ed.), SPECIFICATION CASE STUDIES, Prentice Hall International, Hemel Hempstead, 1988.
5. Duke, R., et al. The object-Z specification language, PROCEEDINGS OF TOOLS 5 (TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS), Santa Barbara, 1991, pp. 465-483.
6. Henderson-Sellers, B. A BOOK OF OBJECT-ORIENTED KNOWLEDGE, Prentice Hall, Sydney (Australia), 1991.
7. Nerson, J-M. Applying object-oriented analysis and design, COMMUNICATIONS OF THE ACM, 35(9):pp. 63–74, 1992.
8. Nerson, J-M. OBJECT-ORIENTED ARCHITECTURES: ANALYSIS AND DESIGN OF RELIABLE SYSTEMS, Prentice Hall, 1993 (forthcoming).
9. Wirfs-Brock, R., B. Wilkerson and L. Wiener. DESIGNING OBJECT-ORIENTED SOFTWARE, Prentice Hall, Englewood Cliffs, NJ, 1990.
10. Meyer, B. Applying "Design by Contract," IEEE COMPUTER, October 1992.

Bertrand Meyer, a SIMULA fan since 1973, is the president of Interactive Software Engineering, Santa Barbara. He is the main designer of the Eiffel language and environment and author of OBJECT-ORIENTED SOFTWARE CONSTRUCTION, INTRODUCTION TO THE THEORY OF PROGRAMMING LANGUAGES, and EIFFEL: THE LANGUAGE (PRENTICE HALL).

ZERO-COST PORTABILITY ≡ ≡ *continued from page 1*

Two major hurdles must be overcome to be successful. The first is to provide a common interface to the underlying operating system capabilities and the second is to provide some mechanism to deal with the problem that not all computers have the same CPU.

The accepted solution to the first hurdle is to employ a virtual machine as mentioned above. The virtual machine provides a platform-independent set of utilities that allow an application program to have uniform access to various operating system components such as the window manager, file system, graphics system, memory manager, and I/O manager. Thus, although the underlying operating systems may be dramatically different, virtual machines make them seem identical to the application program (Figure 1).

To overcome the second hurdle of incompatible CPUs, there are two techniques that employ very different approaches. The classical technique incorporates the use of an interpreter as a part of the virtual machine. In an interpretive environment, the virtual machine includes not only the special layer that transforms operating environments but also a runtime translation system. This virtual CPU converts virtual instructions (the instruction set of the virtual machine) into a series of subroutine calls that implement the semantics of virtual instructions. These subroutines are written using the native instruction set of the host CPU. Using this technique, the source code for the application is "compiled" into the instruction set of the virtual machine rather than the instruction set of any specific processor. The interpreter then performs the function of the CPU by "executing" the virtual machine instructions.

There is a rather severe performance penalty incurred with an interpretive environment. Depending on how closely the virtual machine instruction set matches the actual CPU instruction set, the cost of executing each virtual machine instruction can run from a few to 100 or more CPU instructions. As a result, the application program could execute from one order of magnitude to, in very extreme cases, two orders of magnitude slower than if an interpreter had not been employed.

The second technique was developed to address this performance issue. With this approach, the source code is still compiled to an intermediate form; however, unlike an interpreted instruction, the intermediate form is translated to the actual CPU instruction set at runtime. In the technical literature, this technique is referred to as dynamic compilation or dynamic translation. A more descriptive name would be lazy code generation because the machine code is actually generated as needed and then put into a cache in case it is needed at a later time. As a result of this invention, application programs can be portable yet suffer no serious degradation of performance.

If such a capability were readily available, tremendous savings could be gained by software development organizations that support products on more than one platform. But are we dreaming? Is this really possible?

ParcPlace Systems provides true portability with its ObjectWorks Smalltalk and VisualWorks products. Both products provide a dynamic translation capability that allows programs to execute the native instruction set of the host CPU (Figure 2). Both products provide a virtual machine that presents the operating system utilities to application programs in a completely portable fashion. The architecture of each product is shown in the accompanying figure. Note that both Smalltalk and VisualWorks provide a large selection of reusable components and that both incorporate the Smalltalk virtual machine. The difference between the two products is that ObjectWorks Smalltalk is the standard Smalltalk development environment and VisualWorks is an application development environment that provides database connectivity and a graphical application builder. Both products are available for MS-Windows, Macintosh, most popular UNIX platforms, OS/2, and NeXT.

Product Announcements

Product Announcements is a service to our readers. It is neither a recommendation nor an endorsement of any product discussed.

Softool Corporation

Softool Corporation announced CCC/Manager for Windows and OS/2, a new software change and configuration management product, supporting the development of all object-oriented applications independently of implementation language. CCC/Manager for Windows and OS/2 offers a fully CUA-compliant user interface, permits users to store, retrieve, and manipulate multiple versions of any kind of file, and has an Application Management that enables users to create virtual windows into the repository versions and to update them. Softool's new product operates on networks and stand-alone PCs, without the intervention of a host. When used with host change management systems and Softool's CCC/Bridge product, CCC/Manager for the desktop permits a three-tier architecture covering the development lifecycle. Softool is also now offering a new version of its CCC/Manager product to IBM RS-6000 users.

Softool Corporation, 340 Kellogg Avenue, Goleta, CA 93117, 803.683.3777, fax: 805.683.4105

Virtual Technologies Inc.

Virtual Technologies Inc. introduced its Data Entry Workshop, a collection of tools for writing validated data entry screens and other advanced Windows controls. It's a three step process: Use Resource Workshop to place and edit the controls interactively, run the MAKESRC utility supplied with Data Entry Workshop to generate the source code automatically in C++ or Pascal, and use Borland's ObjectWindows Library to access the controls. Data Entry Workshop is designed for programmers using Borland C++, Borland Turbo Pascal for Windows, or Borland Turbo C++ for Windows. Data Entry Workshop costs $189. No payment of royalties is required.

Virtual Technologies Inc., 46030 Manekin Plaza, Suite 160, Sterling, VA 22170, 703.430.9247, fax: 703.450.4560

Compass Point Software

Compass Point Software completed its five man-year engineering effort with the introduction of application::*ctor* (pronounced "Application Constructor"), a GUI application development tool that relieves the programmer of virtually all the programming burden presented by Windows' graphical environment. The product includes an intuitive, object-oriented "View Editor," a user-interface class library containing over 100 classes, and a C+++ class browser. In addition, application::*ctor* can be used as a very sophisticated prototyping system. application::*ctor* requires Windows 3.1 and your favorite C++ compiler. The product is priced at $99.

Compass Point Software, 332A Hungerford Drive, Rockville, MD 20850, 301.739.9109

Interactive Development Environments Inc.

Interactive Development Environments Inc. has announced the immediate availability of Object-Oriented Structured Design/C++. OOSD/C++ supports development teams doing architectural design, detailed design, implementation, and documentation of C++ applications, as well as creating reusable design components. IDE has initially integrated OOSD/C++ with the ObjectCenter programming environment and with FrameMaker and Interleaf5 technical publishing systems. Integration with SoftBench C++ along with ObjectCenter will be available on the HP platform in the fourth quarter of 1992. OOSD/C++ is available immediately on Sun SPARC-based workstations and servers and will be available on HP 9000 Series 700 and IBM RS/6000 workstations before the end of 1992. Interactive Development Environments Inc. has also introduced the Success Package for C++, which combines IDE's OOSD/C++ with training. IDE consultants work with customers to identify success criteria, define people's roles, set correct expectations, and organize for reuse. The Success Package for C++ is available immediately to accommodate project teams of 5–15 people, with the five-seat package costing $75,000. Although IDE recommends customers start with the Success package, OOSD/C++ is available with one year of software maintenance and technical support and a four-day training class for $10,000 per seat.

Interactive Development Environments Inc., 595 Market Street, 10th Floor, San Francisco, CA 94105, 415.543.9090, fax: 415.543.0145

NeXT Computer Inc.

NeXT Computer Inc. is shipping NeXTSTEP Release 3.0, which preserves all the features of the previous version, while adding improved custom application development tools; greater interoperability;

jects are fully conformant with Object Management Group (OMG) specifications, can be distributed around the office or globe with or without the use of an Object Request Broker, and are completely portable across different platforms and desktops (Fig. 1). Furthermore, these objects can be reused indefinitely in different applications through a simplified version of plug-and-play that is rapidly becoming known as set-and-leave-be.

Developers, regardless of skill level, can become instantly proficient in assembling arbitrarily complex solutions to the simplest of problems. This tremendous productivity boost is achieved by appealing to our childhood play experiences. However, developers who never played with wood blocks in their childhood may experience significant anxiety. When our lab elves compared Amziod Objects with Erector Set products in a compute-off, we found productivity gains of at least 40%. There is no need to bolt components together with the Amziod product, due to its use of a revolutionary new concept for holding things together: gravity. (This may also be an indication that more work gets done when developers treat their work with gravity!)

Our lab test did reveal a few drawbacks, in addition to the minor environmental sensitivities indicated above:

- For starters, the lack of Tinker-Toy compatibility could be a serious problem in winning product acceptance in C++ shops. Our elves suggested that the inclusion of a 3/8-inch drill bit in future releases would solve this problem for all but those large IT network developers who are dependent on routers. The matter of Lego compatibility is slightly more complex and may require the use of sockets.
- Our elves had some doubts about instances being re-entrant once installed in a system. This did not seem to be a problem when enough instances were purchased to supply the needs of all developers, so we would advise site licensing.
- The larger component kits come elegantly packaged in clear plastic boxes that make ideal gifts, but which may cause severe developer trepidations about fitting all the objects back into the box. The manufacturer informed our lab that this packaging is simply intended to take advantage of an efficient form of shipping known as block transfer. Since Amziod plans to introduce new bag packaging (brown paper, if your organization needs to keep your use of objects "under wraps"), this should become a non-issue. (Still a matter of industry concern, however, is why developers should ever want to put objects back in the box!)

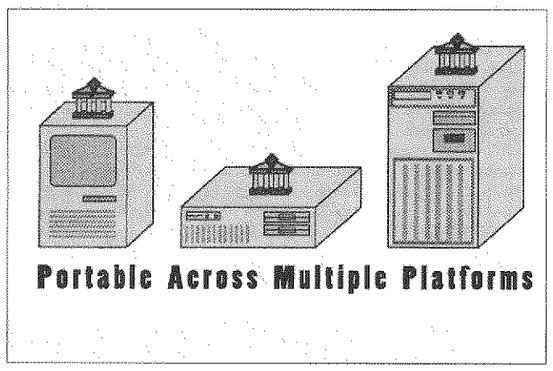


Figure 1. Fully portable.

> ***We think the use of the word 'object' has the potential for increasing sales.***

In all, our lab found the Amziod Objects to be most effective for stress relief, comic relief, vaporware relief, and development. This is simply a blockbuster product that truly delivers everything it promises—and, as the manufacturer explains, when your systems crashes, you can "Just say OOPS."

Every developer needs an Amziod object now!

P.S. From the editor and all of SIGS staff, the best of wishes and good cheer for this holiday season. We hope all our readers have found our reporting and opinions useful during 1992, and we look forward to being of continued service in 1993. ≡≡

Manufacturer:
Amziod
40 Samuel Prescott Drive, Stow, MA 01775, USA
U.S. 508-897-5560 or (fax) 508-897-7332
Technical Specifications:
Made of dark walnut, red oak, cherry, rock maple, ash, and white birch
Supported Platforms:
Any Intel-based X86 system, Apple Macintosh, NEXT, IBM ES/9000 or AS/400, and workstations including SUN, HP and RS/6000
Price Range:
$19.95 to $199.95, or $1.00 per A La Carte object (20 minimum)
Applications:
Great for gifts, especially with your organization's logo on the bag or box

CONCLUSION

Portability is achieved in many traditional development environments at a significant recurring cost—one that is experienced each time a new product release must be ported. Zero-cost portability environments are commercially available today. Where porting cost is a significant financial issue—such as any situation where the cost of porting and providing multiplatform support exceeds the cost of developing and supporting the product on the base platform—development environments that offer zero-cost portability should be evaluated as an alternative. ≡≡

Richard L. Dellinger is Vice President of Engineering at ParcPlace Systems, Sunnyvale, California.

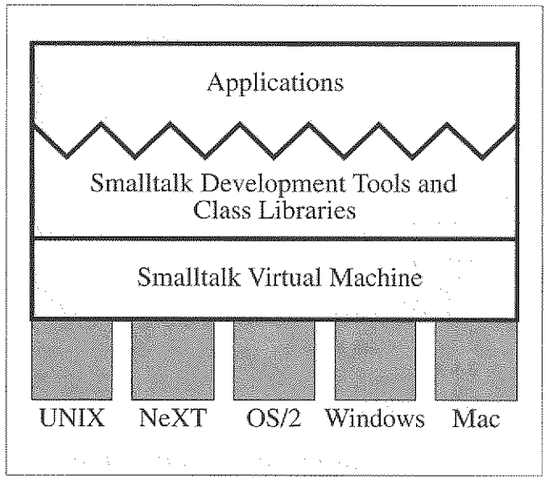


Figure 1.

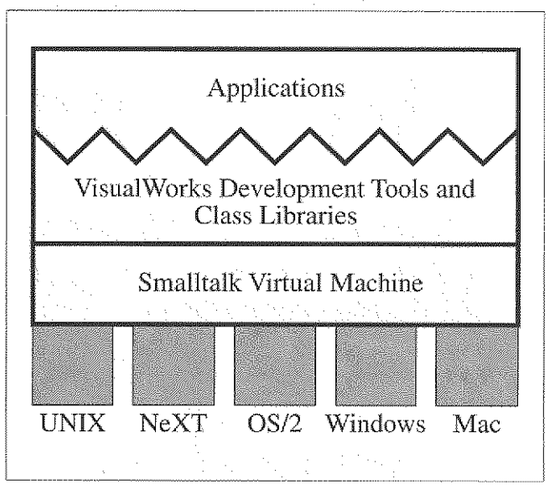


Figure 2.

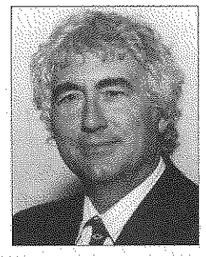
commercial tool builder who supports (or should support) multiple host platforms. Realtime and online transaction processing systems are running today in these virtual machine environments, suggesting that all but the most severe performance concerns hold little merit. The cost of developing and maintaining software in these environments is lower than that of traditional 3GLs. It's high time that developers put aside their prejudices in favor of more cost-effective development tools.

Mr. Bertrand Meyer of ISE writes about design by contract, an approach to designing robust software components. The issues Mr. Meyer addresses are central to successfully selling components and developing CASE tools that will support assembly of applications from purchased components. Component markets like AMIX (see HOTLINE 4[1]) will partly depend, for broad acceptance and high volume business, on effective specification of external characteristics and surrounding usage rules that come with a concept like design by contract. Corporate IT also becomes concerned about object contracts. Consider the trouble that business modelers and data administrators experience when specifying business rules and constraints on such business entities (read that "objects") as customer, financial instrument, or product. The concept of design by contract gives us a mechanism for crisply specifying the behavior, data content, and constraints on business components. Combined with an ER-model–based object model technique, we have a powerful mechanism for specifying business objects in a manner compatible with current modeling approaches.

Mr. Norman Plant, Chairman of the Object Interest Group in England, writes about the next steps his organization must take to further the utility of object technology to large-scale corporate developers in the United Kingdom. This article should serve as a reminder to the object industry that large corporate IT is interested in object technology. We think this user-driven workgroup approach (see Mr. Plant's articles, HOTLINE 3[5], 3[7]) will be increasingly common. Large business IT has learned from experience that more leverage is available in numbers, and is increasingly joining vendor-driven industry groups or forming user-driven groups that focus on specific information technology areas. As Mr. Plant points out, large businesses are fending for themselves when it comes to understanding the integration of object technology into their environments.

Hopefully, the messge is clear: if the object industry will not help, IT will help itself. Object industry, are we listening?

Object interest group: phase two



Norman E. Plant

The Object Interest Group (OIG) was formed in May 1990 and consisted of 16 leading UK companies involved in banking, insurance, manufacturing, chemicals, oil and steel production, telecommunications, and government. The first project it carried out (May 1990–February 1991) was to reach a shared assessment of the technology supported by hard evidence, so that members could understand it, position it in their IT strategy, and tell suppliers what was needed to enable faster exploitation. The project was notable for the spirit of friendly and willing cooperation within the group and the very positive response from the IT industry. In previous issues (HOTLINE 3[5], 3[7]), I have reported on this effort.

Following these encouraging conclusions, during the summer of 1991 members planned to spread their knowledge and findings within their companies to encourage them to start object-oriented projects. A second phase would then be focused on developing method technique and management based on the shared experience of real projects.

TAKE-UP

Take-up within member companies varied widely. Some started immediately with enterprise-critical applications influenced by similar actions of a competitor. Some started by "modelling" to see how effectively it could reflect the changing business of the enterprise. Some started with front-end work and GUIs; in the latter case as a productivity tool for replacing C and PM environments. Others undertook a variety of applications: moving information around the enterprise, process control, client/server. No member used it for their core business high-speed transaction processing systems (e.g., banking transactions, airline reservations, etc). Although it is still early for feedback from these applications, members grow more positive about the technology as they get hands-on experience.

If you are not doing real applications, real problems do not arise and you have no basis for contributing or sharing. We have only recently reached the stage where there is sufficient take-up to provide a platform for our second phase on method, technique, and management.

OIG PHASE 2 MISSION

We defined our mission as helping OIG member companies migrate to object-oriented technology at minimum risk and cost by:

- building confidence on how and where to start
- developing methods, techniques, and management through appropriately focused teams based on sharing real project experiences
- getting the products and services we need to our priorities
- using the power of the group to influence:
  1. the government for funding
  2. appropriate bodies for standards
  3. educational institutions for research and the supply of suitably trained graduates

FULFILLING THE MISSION

We decided the most proactive way of fulfilling our mission would be to identify perceived problems within our companies and form focused teams around them. Interviews with members and their management resulted in the following key concerns:

1. How do I start an O-O project?
2. Where do I start and how do I coexist?
3. What methodology do I choose?
4. How do I manage reuse?
5. How do I assure domain-model quality?
6. How do I support the early days of operating O-O systems?
7. How do I train?

This problem set provides a reasonable focus for each team yet has sufficient overlap to provide issues on which shared agreements must be reached. As teams become more effective, new problems will arise and can be tackled accordingly. Having a problem focus means we are responding to real needs as opposed to studying a topic for interest's sake. Problems 1, 2, and 7 focus on our first mission statement: "confidence in how and where to start." The remaining problems focus on our second mission statement: "developing methods, techniques, and management." We plan to spend nine months on this problem set, with each member company contributing at least 20 person-days. Teams average four people. The scope of the problem areas is discussed below in more detail.

Object-building blocks available today

With so much effort in the object technology industry focused on zero-cost portability, management of reuse, commercialization of off-the-shelf components, and developer productivity, the elves in HOTLINE's product evaluation lab have uncovered what they believe is a significant breakthrough product.

On July 20, 1992, Amziod Corporation announced it was jumping on the marketing bandwagon by renaming an existing product with the word "object." Dennis Merritt, VP of marketing, explained to our product review staff, "We think the use of the word 'object' has the potential for increasing sales." He's right. As pointed out in this issue's "From the Editor," the Information Technology market is substantial, and is looking for products it can get its hands on right away. Since everyone in this industry knows that Amziod's marketing strategy is completely revolutionary and heretofore untried, we simply had to review its product.

Amziod's product is off-the-shelf objects—prefabricated components that can be used to realize an infinite variety of sophisticated structures such as towers, castles, abstract sculptures, engineering marvels with cantilevers, gravity-defying bridges, parquet designs, elegant examples of instability, and whatever else could be inferred into the requirements document.

The manufacturer packages components in various sizes to meet a range of needs. The "Bag of Objects" size is ideal for pilot projects and research efforts exploring the viability of objects, providing 9 base classes and 18 instantiations. Six additional classes (18 more instances!) can be added with the "Objects Upgrade." Individual programmers will be best served by the "Limited Edition CASE of Objects." With 10 subclasses and a full complement of 38 instances, this will keep even the most determined techie busy for days. Our lab confirms that large corporate developers like American Airlines Information Systems (AMRIS) probably would be best served by "A La Carte Objects," a unique approach to packaging in which the buyer can specify exactly the classes and quantities purchased, or Amziod's custom-fabricated objects. Given IBM's troubles in implementing its repository product, and its recent announcement that future versions would utilize object technology, our elves believe IBM should purchase Amziod's "Oodles of Objects." Amziod guarantees that this package of 200 instances from nine base classes will keep an entire mainframe development team busy for weeks. Research is currently under way in our lab to determine if the same effect can be measured in several UNIX development shops—a much tougher challenge!

Though robust enough to serve such mainline production roles as executive entertainment systems, good-luck charms (knock on wood), and US Army Abrams M1A1 tank parking chocks, Amziod's objects are probably best suited for rapid prototyping. Amziod's objects are honed from fine American hardwoods, and as such are both sturdy and aesthetically appealing. For systems being deployed in hazardous or harsh environments, however, some extra precautions will be required above and beyond the manufacturer's Danish oil finish. These objects are not fireproof and months of development work truly can go up in a flash, so smoking while working is strongly discouraged. Also, due to surface porosity, these objects cannot withstand extended contact with coffee or Jolt cola. Beyond these basic cautions, developers should find these objects reliable, stable, well documented, cut to exacting specifications, and at least as much fun as Visual BASIC.

For organizations looking to industry standards efforts to resolve problems with compatibility and portability, Amziod's VP of Development Mary Kroening explained that all Amziod Ob-

These organizations fall broadly into two camps: those with significant investment in existing information systems (usually involving mainframes) and those without a large investment in existing systems. Many of their concerns overlap, while some are unique to each group.

Organizations with substantial existing investment in hardware, software, and networks are almost all in agreement that moving to distributed systems must be a strategic goal. However, they are also uniform in their need to preserve or, better yet, enhance existing investments during the transition. These organizations are being driven today by user demand for more access to existing systems from graphical interfaces running on desktop machines. Consequently, issues around client/server, where the server is a host system, and GUIs are the most important topics today.

In contrast, organizations with less existing investment are more concerned with building new applications rapidly and being able to distribute the processing of these new applications in a very flexible manner. Issues of primary concern center around peer–peer distribution between powerful workstations and local servers, and tools for building applications that can operate in this peer–peer environment.

Given these business goals, we can examine technologies in terms of their capability to help organizations move toward their distributed processing goals; in particular, how object technology can help. Object databases, for example, are being used today to provide distributed processing. Another example is the Object Request Broker from the Object Management Group, which is a promising vehicle for trading objects across different hardware and software environments. There are also development environments that allow rapid delivery of GUI-based applications using object programming languages. In future columns I will discuss these and other object technologies that can assist organizations in achieving positive ROI via distributed processing of information.

CONCLUSION

Return on investment must always be the ultimate criterion for judging the effectiveness of software. ROI forces the explicit examination and statement of business goals, which can be used to evaluate an organization's investment in technology. Object technology holds great promise in providing business solutions through distributed information processing. In future columns I will examine various object technologies and how they can contribute to improving business processes using distributed information processing. In addition, I encourage you to let me know what issues are most relevant to your organization. I will do my best to cover those issues most relevant to our readership. ≡≡

Tim Andrews is Chief Technical Officer of ONTOS, Inc., and may be reached at ONTOS, Inc., Three Burlington Woods, Burlington, MA 01803, by voice mail at 617.272.7110 x288, or via email at andrews@ontos.com.

PROBLEM AREAS IN DETAIL

How do I start an O-O project?

The deliverable is the ability to stand up in front of your management and project the confidence that you know how you are going to start and run an O-O project. Its scope includes project and quality plans, commercial and technical objectives, documented inputs, intimate domain-knowledge inputs, facilitating a modelling team and keeping the process going when the team gets stuck, recognizing emerging model stability and when to stop, object-oriented questioning, project stages, roles, deliverables, reporting points, etc.

> ***If you are not doing real applications, real problems do not arise and you have no basis for contributing or sharing. We have only recently reached the stage where there is sufficient take-up to provide a platform for our second phase on method, technique, and management.***

Where do I start and how do I coexist?

The purpose is to enable members to look at a mainstream application in their company, advise on where object-oriented technology can be introduced with benefit, say how key technical issues (e.g., coexistence) can be handled, and provide supporting evidence from external projects.

Its scope includes:

- strengthening relationships with external practitioners to gain access to hard evidence
- studying where people have started
- looking at how detailed technical issues were handled (e.g., "front-end" objects sharing attributes with legacy-system databases, front-end systems integration, front-end O-O databases coexisting with mainframe relational databases, etc.)
- drawing out general guidelines

What methodology do I choose?

This project must cover all aspects of the two interacting lifecycles (the domain component lifecycle and the application lifecycle). Most current methodologies do not recognize the difference between domain and application and therefore do not support reuse.

The scope includes looking at available and emerging O-O methodologies, their strengths and weaknesses, what people's experiences have been, coexistence with installed conventional methodologies, migration from existing methodologies, enabling software tools, training and skill requirements, management issues, standards, consistency of notation from analysis to code, absorption of external foundation classes, etc. The deliverable is the inventory and evaluation of O-O methods and their potential impact on current working practices, techniques, and procedures.

How do I manage reuse?

The scope covers visibility and understanding of the range of libraries beng made available from internal and external sources and the management issues associated with their use. Library types include generic system objects such as GUIs, generic applications such as payroll, and the forms in which they are available (code or just design).

Reuse management issues include organizational infrastructure, corporate architecture, external library assessment, identifying parts of applications that are candidates for libraries, library management, enabling tools, standards, ensuring reuse, and handling the associated cultural issues.

How do I assure domain-model quality?

Quality comes from the reuse of tried and tested procedures, designs, components, etc. The purpose of this project is to establish a set of guidelines or designs for common business objects that frequently occur and can be recognized in many situations. "Business Party" (a person or organization with whom we interact) and "Business Relationship" (the nature of the relationship between ourselves and the Business Party) are two such objects. Both have subclasses along the lines of person, company, and internal department. The idea of an object "Business Relationship" seems odd in the conventional entity-relationship approach, but in this generic form it effectively handles suppliers who are also customers, internal company dealings, etc. It is important for the analyst to have an inventory of good design practices if domain quality is to be assured.

The approach includes:

- understanding to what extent extent generic objects can be designed that are widely applicable to the internal working of most companies. Examples include Business Party, Business Relationship, Budget, Client Perception, Organizational unit, Plan, Resource, etc.
- understanding the extent to which sound generic objects can be designed that are widely applicable to industry types, in particular to their products and product distribution (e.g., separate foundation classes for airlines, banking, insurance, etc.)
- identifying existing and potential sources of supply
- supplying visibility in the form of catalogues, electronic or hard copy, and detailed description guidelines
- understanding the competitive-edge issues
- assessing if the object set defines a sufficiently complete language to meet most application needs

How do I support the "early-days" operation of O-O applications?

As yet we have not given the project adequate definition but it concerns those member companies cutting over their first projects and having to maintain and operate them with very little in-house experience.

Among the issues expected to be pertinent as applications are cut over to production are:

- application deployment and hardware/software configuration-management issues
- security and access control at all levels of the architecture
- troubleshooting defects
- tuning application performance

How do I train?

The OIG has produced a two-day workshop based on the findings from our first project (May 1990–February 1991). The workshop is a far more effective way of communicating widely than are reports from the same project. This workshop will become the repository for recording and communicating solutions and any future problem areas. By this means, the findings of our group can be effectively fed back to IT divisions. Even though member companies are active with the technology, penetration and understanding across IT divisions as a whole are still very slight.

The scope of this project obviously includes understanding the approach to training by external suppliers and assessments as well as comparisons of their training products and services.

HARD EVIDENCE

Real experiences are essential to these problems; thus the approach is to get hard evidence from UK, USA, European, and our own member companies (users and suppliers). In return we will share current and prior findings with any company willing to help. This worked well on our first project and proved that establishing these relationships is an important task.

THE BUSINESS CASE

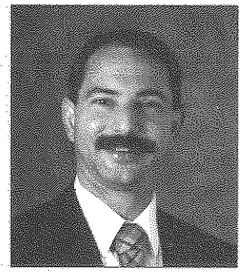
Each of our member companies will contribute a minimum of 20 person-days and share solutions to all problem areas. They will have international visibility of their application and use of object orientation. They also will have a social network of people they can easily contact. Non-member companies who can contribute real and relevant experiences also may share in our findings.

CONCLUSION

I hope the second phase will have the same measure of success as the first. Industrial collaborations are not easy, particularly when members are competitors. But good progress can come from wide-scale interaction of committed, focused minds, and we need to understand the competitive issues associated with this technology. Should any other group wish to start on similar lines, we would be only too happy to help. ≡≡

Norman Plant is Chairman and a founder of the Object Interest Group and a consultant specializing in the inroduction of object technology into large IT organizatios. He was a senior manager in British Airways' IT division for 25 years. He can be reached at +44.252-836315 in the UK.

Towards a framework for software ROI



Tim Andrews

In my last column (HOTLINE 4[1]) I discussed the need for approaching software evaluation from an economic perspective, focusing on return on investment (ROI). This column will continue that subject by working towards a basic framework that organizations can use to implement ROI analysis.

THE PROCESS IS THE GOAL

The most important consideration for an organization preparing to undertake an ROI analysis is that the process is ongoing. It is certainly helpful to conduct some initial ROI work, but the real benefit is derived from the establishment of a process that feeds back into the system (Fig. 1).

The process of ROI analysis will start as a process of discovery: How is software evaluated? What cost items are included? Once a selection has been made, the process continues: What is the cost of development relative to estimates made during the evaluation? What are the causes of any variances discovered? Finally, are the costs of deployment and maintenance as expected? And is the value provided by the software as expected? This is where the "R" in ROI can be evaluated.

As an organization gains experience with this process, a larger picture emerges. The organization's true costs are identified, along with actual returns of various software systems. This information enables the organization to more accurately predict both costs and ROI of new software development, which in turn leads to better budgeting ability and the IS equivalent of portfolio management: how to maximize ROI.

When ROI analysis is used, organizations begin to consider risk vs. reward, which is a very important and positive change. In the absence of ROI analysis, organizations tend to focus on avoiding risk because risk is easily recognized. Reward is usually based on an abstract future goal and is not always easy to quantify, so this factor tends to be ignored. Thus a software product that offers very high reward is often rejected because it also has significant risk. Even if the reward is great in proportion to the risk, organizations may not choose a software technology because they cannot compare the risk/reward ratio to that of other software. When ROI analysis is adopted, a "Software Value Graph" can be drawn, which represents the comparison of software based on risk-reward characteristics (Fig. 2).

As more analysis accrues, the graph is updated. The graph provides insight for the organization about the value of chosen software in its own environment. Organizational differences such as existing hardware and software infrastructure, training and education of personnel, and competitive environment affect the ROI of a software system, which in turn affects the organization's risk/reward profile. For example, an organization with large batch processing operations on mainframes would perceive a different risk/reward from software available on a distributed workstation than would a smaller company that runs on distributed workstations. This difference in perception does not mean that the software is more valuable to the smaller organization, but that ROI analysis will be different for these two organizations.

DISTRIBUTED INFORMATION SYSTEMS AND ROI

I want to use these initial columns to establish an ROI-based perspective from which to discuss information systems; in particular, distributed information systems. In this way, I hope to be a "user advocate" by consistently focusing on the ultimate economic benefit, or ROI, of a given approach. Now I want to begin the examination of distributed information systems and their impact on organization.

Below I will outline those areas I have found to be of concern to the organizations I have talked with over the last several months.

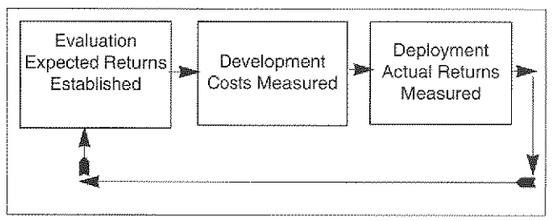


Figure 1.

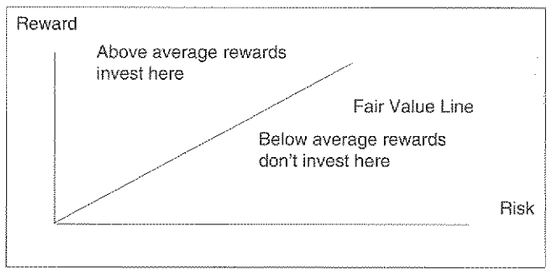


Figure 2.

hotline on
OBJECT-ORIENTED
technology

Back issues

All back issues of the HOTLINE are available. Please call 212.274.0640 for details.

Vol. 4, No. 2/December '92 ≡ Zero-cost portability ≡ Design by contract: Builing bug-free O-O software ≡ Object interest group: phase two ≡ Towards a framework for software ROI ≡ Reviewing Amziod "objects"

4, No. 1/November '92 ≡ Combining object technology with data standards for the next industrial revolution ≡ Constant quality management ≡ Evolving markets for software components ≡ The quest for value ≡ Reviewing OOSE: a use case-driven approach

Vol. 3, No. 12/October '92 ≡ ROI: development environments for the lifecycle ≡ Selecting the right object-oriented method ≡ Choosing an object-oriented language ≡ Object database technology: who's using it and why? ≡ Objects and reuse

Vol. 3, No. 11/September '92 ≡ Developing strategic business systems using object technology ≡ Object training: harder than it looks ≡ Object-oriented ROI: extending the CRC across the lifecycle ≡ What TQM means for OT

Vol.3, No.10/August '92 ≡ Object technology: toward software manufacturing ≡ Return on investment: software assets and the CRC technique ≡ Object-oriented technology in Japan ≡ Providing commonality while supporting diversity

Vol.3, No.9/July '92 ≡ OOD: Research or ready ≡ Enterprise modeling: an object approach ≡ OMG's 18–24 month view ≡ Design for object-oriented applications: a CASE for wishful thinking . . .

Vol.3, No.8/June '92 ≡ Business in the Information Age ≡ From data modeling to object modeling ≡ How frameworks enable application portability ≡ Interview with Vaughan Merlyn

Vol.3, No.6/April '92 ≡ Thinking the unthinkable: reducing the risk of failure ≡ Mitigating madness with method: first establish what you value ≡ Championing object technology for career success in the 1990s ≡ Objects and actions in end-user documentation

Vol.3, No.5/March '92 ≡ TA large-scale users' assessment of object orientation ≡ Report on the Object-Oriented COBOL Task Group ≡ Interview with K.C. Branscomb

Vol.3, No.4/February '92 ≡ The big prize: acceptance of O-O by the MIS community ≡ Retrospective: 1991—the year it all changed ≡ Making the transition to O-O technology ≡ Interview with Beatriz Infante

Vol.3, No. 3/January '92 ≡ Enterprise object modeling: knowing what we know ≡ Adopting objects: pitfalls ≡ Adoption rate of object technology: a survey of NSW industry

Vol.3, No. 2/December '91 ≡ Accepting object Technology ≡ Adopting objects: a path ≡ Incorporating graphical content into multimedia presentations

Vol.3, No. 1/November '91 ≡ Leading the U.S. semiconductor manufacturing industry toward an object-oriented technology standard ≡ Coping with complexity: OOPS and the economists' critique of central planning ≡ Choosing Object Technology: What's the object? ≡ OOP: the MISsing link

Vol.2, No. 12/October '91 ≡ A modest survey of OOD approaches ≡ What is a "certified" object programmer? ≡ Perspective: investing in objects today ≡ Object oriented in Melbourne, Australia ≡ The Object Management Group

Vol.2, No. 11/September '91 ≡ From applications to frameworks ≡ Report on the Object-Oriented COBOL Task Group ≡ Getting started with object technology: efffectively planning for change ≡ Object statistics on the way ≡ On objects and bullets

Vol.2, No. 10/August '91 ≡ Distributed object management: improving worker productivity ≡ Getting the best from objects: the experience of HP ≡ APPLICATIONS: EC employs object technology ≡ CAPACITY PLANNING: Fiddling while ROMs burn

Vol.2, No. 9/July '91 ≡ Multimedia is everywhere! ≡ Developing an object technology prototype ≡ Object-oriented capacity planning ≡ How OOP has changed our developmental lifecycle ≡ Modularization of the computer system

Vol.2, No. 8/June '91 ≡ Domain of objects: the Object Request Broker ≡ Object-based approach to user documentation ≡ Report on the Object-Oriented COBOL Task Group ≡ Do we need object-oriented design metrics?

Vol.2, No.7/May '91 ≡ Hybrid object-oriented/functional decomposition for software engineering ≡ So, what makes object databases different? (Part 4) ≡ Using the generic application to solve similar domain problems ≡ Experiences using CLOS ≡ International Conference on Object-Oriented Technology, Singapore

hotline on
OBJECT-ORIENTED
technology

VOL. 4, NO. 3 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES JAN. 1993

Gauntlet implementations: the 5GL object-oriented challenge

Bernadette Reiter

Without doubt, computers and their related technologies—hardware, software, communications—have had the most powerful impact on the second half of the twentieth century. In a few short decades, man has moved from subsonic flight to visiting the moon. The secrets of DNA and RNA have been revealed, and global news and trading systems have shrunk the vastness of the five continents to the immediate interactivity of a medieval village. In all these achievements, and in many others, computers and computing have played major roles. Their central importance in our daily lives can scarcely be exaggerated.

And yet exaggeration there has been and continues to be. Despite the almost incredible achievements made using computers and software products, they still often fail to live up to the claims made by their vendors. This is particularly so in commercial environments where, despite many real advances, boasts and promises still frequently outstrip performance, often by a very large margin. Almost daily, newspapers and magazines carry articles reporting that large systems have been abandoned or failed. A casual browse through recent, well-publicized press reports shows the abandonment of $1 billion development in travel-related systems at AMRIS; another of over $200 million for the rewrite of financial systems at AMEX; an undiscovered (until years after the event) $40 million alleged fraud in building an $80 million hospital system at Wessex Health Authority, England; the total abandonment of a custom-built ambulance dispatching system that could not dispatch ambulances at the London Ambulance Service; and the list could go on . . .!

HISTORICAL PERSPECTIVE

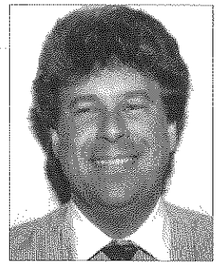
It is worth briefly reviewing how the information technology (IT) industry came to this current state of affairs. Data processing, or IT as it is now usually called, has undergone massive changes and improvements. While hardware price and size have gone down, performance and, more notably, functionality have increased by a factor of 1,016. However, over the same time period software implementations have only evolved in speed of implementation, price, performance, and functionality by a single factor of 101. In addition, it is generally believed that half of this relatively tiny "leap" occurred when developers migrated from programming in second-generation assembler languages to third-generation languages (3GLs) such as COBOL, FORTRAN, RPG, etc., something that took place three decades ago (for everyone except the airline reservation systems . . .)!

Hardware has achieved its dramatic accelerated evolution through the ongoing development of building blocks. This

continued on page 4

IN THIS ISSUE

FROM THE EDITOR



One year ago my first article for the HOTLINE was about enterprise object modeling. Today, at Open Engineering, we are partners in the Integration Consortium, where we are engaged in the final throes of producing a (hopefully pivotal) paper on the subject for presentation at OOP '93 in Munich. In rereading my original article, and some of my early editorials, I was bemused by signs that this old dog may have learned a few new tricks. Many of these, by the way, are thanks to colleagues like Mr. Thomas Bruce at Bank of America; Ms. Kate Salafia and Ms. Elizabeth Sevean of the Integration Consortium; Mr. Norman Plant, a frequent Hotline author and Chairman of the Object Interest Group in London; and Mr. William Perrin, a client of some years back who has since gone into business to develop innovative, money-saving software solutions for the commercial insurance industry. He's using object technology, of course!

My concerns for the object technology industry revolve around presenting an accurate vision and useful road map of object technology to the Information Technology (IT) community. I am also gravely concerned about the viability of industry in this country, and the ability of IT to support and rejuvenate its competitiveness. My above-mentioned colleagues have helped hone my awareness of the issues and concerns facing large IT organizations, and have helped me focus on what makes software truly useful to business. As my second year as HOTLINE editor begins at a fast trot, I would like to take this opportunity to thank these individuals, and others here unnamed, who have given their valuable time to educate, coach, counsel, cajole, and generally keep my feet on the ground. This, I think, is what the camaraderie of a healthy business community is about. This I would like to see us extend to the global community in the months and years to come.

I would like to take this opportunity to introduce three new Editorial Review Board members who have joined us in recent months. Ms. K.C. Branscomb, whom you will remember from our dynamic interview (HOTLINE 3[5]), is former CEO of Intelicorp, and currently Senior VP of Business Development for Lotus Development Corporation. Congratulations to Ms. Branscomb—and to Mr. Manzi, as well. As you will recall from our interview, Ms. Branscomb has argued that no development tool will win with the IT community until it addresses their needs and integrates their legacy systems. This is an insight with which many O-O product companies have been grappling during 1992, and I hope to see the fruits of their labors in this new year. We look forward to Ms. Branscomb's continued involvement in the software development community, and to her advice and counsel here at the HOTLINE.

Ms. Bernadette G. Reiter, CEO and President of Objective: Inc. and our feature author this month, brings to our Board her self-professed unconventional perspective on software development and the role of software in business. Perhaps it's my own leanings on the subject, but Ms. Reiter's views on cost-effective development, cut-through-the-bull methods for delivering really creative applications, and conviction that software must be put back in the hands of the business end-users to give them back control over the business all make sense here! The challenge at hand is to show business end-users and IT professionals that alternatives to today's morass do exist, and that they really can pick up from where they are today and make forward progress. This is certainly a challenge that Ms. Reiter and Objective: Inc. have taken on.

Mr. John A. Zachman, formerly a Senior Consultant with IBM, and presently a highly respected consultant to the IT community through his firm Zachman International, brings to our Board his perceptiveness about large IT organizations and the underlying problems we are trying to solve. Mr. Zachman is known throughout the IT community for his work on the Framework for Information Systems Architecture.[1,2] The Framework provides a basis for understanding the business and systems development process that recognizes the need for understanding concurrent perspectives on a business system (those of planner, owner, designer, builder, and subcontractor). That methodologies to date have not accounted for these concurrent views explains part of IT's current predicament. Mr. Zachman's writing is thought provoking and highly recommended. We welcome him to our Board.

Turning our sights to the year ahead, I would like to share my editorial plan with the hope that feedback and improvements will be forthcoming from our readers. We will devote at least four issues to substantial real-world applications of object technology; currently March, May, July, and September. The April issue will focus on reuse. Other issues will feature client–server development tools, database and legacy systems encapsulation, middleware and object request brokers, and some fresh thinking in engineering methods and support tools. Let me know what you think.

Let's take a look at this issue. Our feature article by Ms. Bernadette Reiter argues for 5GL development environments as an alternative to conventional programming languages. What underlies Ms. Reiter's case for retooling business application developers, is a value proposition: break the techno-cultural taboos about "right" and "wrong" development tools. These taboos are rooted not in current fact, but in historical familiarity and bias. As IT organizations look to become "object oriented," familiarity draws them to Object-COBOL, C++, and Objective-C. Business application developers *can* use these tools, but Ms. Reiter's point is that they don't *have* to. Instead, use the right tool for the job. Total cost and time to market argue against using raw language tools for most business application development, especially as our concept of user interface moves beyond text fields in windows to visual models like maps and diagrams.

When evaluating 5GL tools, here are some requirements to keep in mind: Does it support connectivity to the database management systems in your environment, or can such connectivity layers be installed that give the developer an object view of the database? Is client–server architecture sup-

DISTRIBUTED SYSTEMS . . . Users clearly see the spread of powerful desktop workstations occurring, and the resultant move to put applications where users are located rather than in a central computing center. The hot issues for the next few years will be whether and how much to move from a centralized to a decentralized computing and communications architecture, and the increasing complexity of managing hybrid environments. This complexity also seems to underlie the [1992 Network and Distributed Systems Management Conference] attendees' preference for object-oriented approaches to management applications. There is a strong belief that object-oriented approaches will both accelerate application development and increase interoperability among applications and objects . . . The clear message to standards organizations and vendors was that platforms and infrastructure are great, but only if they help get applications with increased flexibility and interoperability on the market sooner . . .

Net Management Directions, James Herman, BUSINESS COMMUNICATIONS REVIEW, 8/92

LANGUAGES . . . Although no C++ compiler is available on the AS/ 400 system, many AS/400 programmers are purchasing C++ compilers for their PCs. I frequently am asked whether it is necessary to learn C before learning C++. My answer is not. Knowing C definitely will help you learn C++ because the languages are so similar, but in my opinion, such knowledge is not a requirement. A good analogy is that although knowing how to drive a car with an automatic transmission makes it easier to learn to drive a standard-transmission car, I wouldn't recommend learning to drive an automatic just to learn to drive a standard. Also, because C++ is a separate language, most C++ reference books treat it as such and cover the whole language, not just its enhancements over C . . .

C++ for AS/400 programmers, Jennifer Hamilton, NEWS 3X/400, 9/92

Book Watch

Waite Group Press, 200 Tamal Plaza, Corte Madera, CA 94925, 415.924.2575; fax: 415.924.2576

OBJECT-ORIENTED PROGRAMMING IN MICROSOFT C++ by Robert Lafore ($29.95) takes you into the newest techniques of object-oriented programming using Microsoft's C++ compiler and environment. You will find complete, step-by-step lessons on the basics.

BORLAND C++ DEVELOPER'S BIBLE by Mark Peterson ($29.95) goes beyond Borland's own documentation and provides references and tutorials for the commands and options found in the Borland C and C++ compilers, linkers, debuggers, assembler, libraries, and utilities. It also introduces and references the new Applications Framework including Object Windows.

Prentice Hall, Simon & Schuster Education Group, 113 Sylvan Ave., Route 9W, Englewood Cliffs, NJ 07632, 201.592.2348

OBJECT-ORIENTED PROGRAMMING by Peter Coad and Jill Nicola ($33.00) is the third book in a series on object-oriented development. Through a series of four comprehensive examples written in Smalltalk and C++, this book teaches the reader how to "object think": up-front concise thinking and conceptualizing with object-oriented development.

TRANSFORMING THE ENTERPRISE THROUGH COOPERATION: AN OBJECT-ORIENTED SOLUTION by Dan Shafer and David Taylor ($15.00) provides a concise overview of NCR's full-scale enterprise computing systems based on object technology and how it can be used to help manage an organization.

TOOLS 7, Bertrand Meyer, George Heeg, and Boris Magnusson ($60.00). Proceedings from the TOOLS 7 Conference held in Dortmund in April, 1992. TOOLS 8, Bertrand Meyer, Riamound Ege, and Madhu Singh ($60.00). Proceedings from the TOOLS 8 Conference held in Santa Barbara, CA in August, 1992.

F.Y.I.

Excerpts from leading industry publications on aspects of object technology

MODELING

. . . Clearly, we do not suffer from any shortage of models in our companies. In fact, the problem is just the opposite — we build too many models, and we build them all independently and incompatibly . . . With so many disparate but overlapping models and methodologies, changing any one model is bound to have unpredictable consequences for all the other models . . . A fundamental tenet of the object-oriented approach is that all aspects of real-world objects should be encapsulated into their corresponding software objects. This tenet is most explicit with regard to data and operations: A well-designed object contains all the data and operations it requires to represent its real-world counterpart. This encapsulation of data and operations folds the corresponding data and operational models into a single object model. It also allows changes in the object model to automatically affect both data and operations in a consistent, symmetrical manner. Since there is only one model for data and operations, any change in this model automatically keeps the two consistent with each other. With a modest amount of extrapolation, we can readily see how all the other models can be rolled into a unified object-based model as well . . . As for simulation, it almost comes with the territory . . . We simply copy the current state of the model, make whatever modifications we would like to explore, and then "run" the model to discover the results. In effect, we can play "what if" with the entire company rather than with a spreadsheet full of questionable numbers . . .

Easing into objects: Integrating the enterprise through object modeling, David A. Taylor, OBJECT MAGAZINE, 11-12/92

STRATEGIES

. . . [ParcPlace's Adele Goldberg:] We talk to people doing mission-critical applications, and they get this message that life is fun, life is easy, that you can ignore all the complexity, that it all happens for you — all because of objects. Well, it's not because of objects. What we're saying is that we can provide the know-how that allows you to specialize and refine and solve your particular needs within a predefined application space. But if you move outside that space, which is what you do when you do mission-critical work, you have much bigger issues to contend with. The MIS staff that builds these system knows this, and now they have to confront an end-user community that believes the marketing messages that say object are the answer and make it all easy. The good news is that the 1990s are about application developers, about users working more as a team. There's an opportunity here, even if the hype sometimes gets in the way . . .

Making object-oriented smalltalk, **DBMS**, 10/92

. . . Object wrappers can be used to migrate to object-oriented programming while protecting investment in conventional code . . . . It is possible to create object wrappers around the bulk of existing code that can then be replaced or allowed to wither away. Building object wrappers can help protect the investment in older systems during the move to object-oriented programming. An object wrapper enables a new, object-oriented part of a system to interact with a conventional chunk by message passing. The wrapper itself is likely to be written in the same language as the original system, let us say it is COBOL, for example. This may represent a very substantial investment, but once it is in place virtually all maintenance activity may cease; at least, that is the theory . . .

Migration strategies: Interoperation—combining object-oriented applications with conventional IT, Ian Graham, OBJECT MAGAZINE, 11-12/92

APPLICATIONS

. . . I cannot imagine that there are a whole lot of people — in the Silicon Valley region or anywhere else, for that matter — who have the skills to do factory automation programming and who are spending their time working as assemblers. [NeXT's CIM Manager Steve Herrick] admits "Our operators don't know how to program." Which might seem to put them in awkward positions vis-a-vis their responsibilities. The trick to their doing the programming is the NeXTstep software system . . . Consequently the operators can do the necessary programming by entering the particular processing parameters, then hitting the "Go" icon. The work is done for them by the system itself..

What's NeXT? Gary S. Vasilash, PRODUCTION, 9/92

ported, and is the networking transparent to the application? Is the application portable without change across host platforms and GUIs? Is the environment extensible through componentization? How is team development and configuration management supported? How is sharing critical business objects handled? Can the vendor (or a third party) provide connectivity to request broker and transaction processing layer products?

Speaking of distributed computing, Dr. Robert Marcus of Boeing Computer Services writes about middleware: the glue that enables more powerful and sophisticated connectivity, distribution of services, and interaction among software components. Anyone familiar with an online transaction processing (TP) environment will recognize TP layers as middleware products. The ultimate goal of middleware is full-bore distributed object computing: enterprise-wide sharing of a variety of resources from business objects to spell checkers to CPU cycles. Middleware can make available on workstations throughout the enterprise services that would otherwise be confined to the application that delivers them. In addition to distribution and connectivity, middleware offers interoperability through standard application program interfaces (API), portability and platform independence, a plug-and-play platform that will encourage the growth of the software component industry, and a basis for intervendor product innovation and user-constructable applications. Middleware is the enabling technology for the distributed computing architecture that, as Mr. John Rymer of the Patricia Seybold Group observes[3], will predominate through the end of this decade.

Mr. Sergio Simmel of the Penobscot Development Corp. writes about software licensing and metering. Meterware, or pay-per-use software, is the most sensible approach to resolving the complexities of licensing where software components are involved. From the business end-user's point of view, pay-per-use makes sense over flat-fee licensing where the use of a particular software component or application is infrequent. Consider how metering removes both the motivation and mechanism for software piracy when, say, a developer needs to use a USD $10,000 CASE tool for a week or two. Consider, also, the cost savings: free distribution (copy and use), no lengthy justification process to management for the purchase of a costly tool, and rapid access to the right tool right now. As Mr. Simmel points out, pay-per-use is the sign of a maturing marketplace.

Mr. Christopher Keene of Persistence Software proposes a very practical entreé to objects for IT organizations: start with products that give developers an object-oriented view of existing shared data structures. I have long maintained that the O-O development tool that cannot integrate with legacy systems and databases using off-the-shelf connectivity is of questionable value to IT. Furthermore, it is unlikely to be taken seriously for production development, because, as Mr. Keene points out, new applications usually have to access today's corporate data where it lives. By the way, Mr. Keene's product is the kind of solution that can make C++ sane! Combined with application frameworks like Zapp from Inmark Development, there is yet promise for shops dedicated to C++: developers can rise above machine-level coding and focus on the business application at hand. This is particularly critical because most middleware products still require a C or C++ interface when accessed from a 5GL application.

In contrast to relational database technology, columnist Mr. Timothy Andrews undertakes to answer the question "What is an object database, and what is it good for?" I expect readers will find his next several articles very helpful in this regard, especially because, as Mr. Keene points out, both the RDBMS and ODBMS have their places in the business and technology picture today.

The right tool for the job will be the watchword for 1993.

References

1. Zachman, J. A. A Framework for information systems architecture IBM SYSTEMS JOURNAL 26(3):276–298.
2. Sowa, J.F. and J.A. Zachman. Extending and formalizing the framework for information systems architecture, IBM SYSTEMS JOURNAL 31(3):590–616.
3. Rhymer, J. R. Distributed object computing, DISTRIBUTED COMPUTING MONITOR, 7(8): August, 1992.

hotline on OBJECT-ORIENTED technology

Robert Shelton, Editor

SIGS Publications, Inc.

Richard P. Friedman, Founder & Group Publisher

ART/PRODUCTION

Kristina Joukhadar, Managing Editor
Susan Culligan, Pilgrim Road, Ltd., Creative Direction
Elizabeth A. Upp, Production Editor
Jennifer Englander, Art/Production Coordinator

CIRCULATION

Ken Mercado, Fulfillment Manager
Vicki Monck, Circulation Assistant
John Schreiber, Circulation Assistant

MARKETING

Amy Friedman, Projects Manager
Lorna Lyle, Promotions Manager—Conferences
Sarah Hamilton, Promotions Manager—Publications
Caren Polner, Promotions Graphic Artist

Administration

David Chatterpaul, Bookkeeper
Ossama Tomoum, Business Manager

Margherita R. Monck, General Manager

Jane M. Grau, Contributing Editor

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012, (212)274-0640.

Mailed First Class. Subscription rate — one year (12 issues) $249, Foreign and Canada $279. Single copy $25.

POSTMASTER: Send address changes & subscription orders to HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville, NJ 07834.

Submit editorial correspondence to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123 voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of HOTLINE ON OBJECT-ORIENTED TECHNOLOGY, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, THE X JOURNAL, C++ REPORT, THE SMALLTALK REPORT, and THE INTERNATIONAL OOP DIRECTORY.

started with clocks and digital logic, moving quickly to flip flops, integrated circuits, larger components, chips, and boards. Each evolution produced more powerful building blocks that were available to build a new computer starting from foundation logic. This new approach produced a consistency and robustness of products while dramatically reducing development cost and time.

In a similar way, software evolved from early machine code that forced developers to program in the numeric equivalent of the instruction set. This layer gave rise to assembler languages that allowed programmers to create applications in a symbolic representation of the equivalent machine code. This layer was then used to create third-generation languages that supported "high-level" computing constructs (e.g., total=quantity * price), which then converted to many lines of assembler language code. Third-generation languages and their predecessors are used not only to describe the logic of an application but also act as an integrator of technologies (e.g., integration of database managers, network drivers, terminal emulators, business graphics, statistical packages, etc.).

Fourth-generation languages (4GLs) were an anomaly. They arose from the database community allowing nontechnical developers to implement ad hoc reports and simple transaction interfaces.

Although each "generation" of language evolution appreciably improved productivity and moved technologies closer to the end user, none has produced the breakthrough "implementation language"—the descriptive symbology—that revolutionized the development of hardware. For example, the concept/symbol/functionality of a math coprocessor, as opposed to the tens of thousands of gates and low-level "components" needed to define it from ground zero. The first four generations of software language have not produced a similar breakthrough in their plodding evolutionary process. Programs written in Ada, C, FORTRAN, Modula X, or Zeetalk are still hand built, tediously and repititiously created from the equivalent of the earliest building blocks of the third generation. Because of this, these creations are inevitably slow to build, liable to failure, and difficult to change. Again in hardware terms, there is no capability for identifying the faulty board and swapping it out for a new one.

THE 5GL ALTERNATIVE

The obvious questions at this point are, "Surely there is a fifth-generation language (5GL) around the corner and, if there is, what is it and when will it arrive?" Quick answers to these are, "Yes there is; it's almost certainly based on 'object-oriented' approaches; and it may be here already." These answers certainly need expansion!

Firstly, what is a 5GL? Is it even a goal worth pursuing? This is not an idle question as the phrase itself has already been debased by premature use. A decade ago, the search for the "holy 5GL Grail" was very fashionable. Japan, under the prestigious auspices of MITI, sought to achieve dominance in software by committing billions of yen a year for "as long as it takes" to create a 5GL. To date, little has been achieved and even less heard about it for the last five years or so. The target was, in somewhat oversimplified terms, to build a "language" that would enable a human to talk to a computer and have it do his/her bidding. Significantly, the phrase "object oriented" was never mentioned.

Object-oriented approaches today appear to offer the most likely bases for 5GL environments. The concept of objects has been around for a long time but only recently have they been touted as the new panacea for building computer software. The concept of objects is simple. An object is a reusable "piece" of code with a specific operation or functionality. Many objects can (should be able to) be quickly bolted together to form an application. New applications can be built much in the same way as hardware. Objects can "inherit" features from other objects. The goals are: speed of implementation, modularity, reusability, and robustness. The concepts are simple, the goals laudable, but . . . once again, achievements lag well behind publicity.

Perhaps the key reason for this is a misunderstanding of what an object should be—what precisely is the system builder trying to modularize and reuse? It can be argued that today most object-oriented approaches focus on the wrong building blocks. For example, many recent articles on object-oriented technology highlight its components as noun-based examples and analogies (e.g., a house inheriting a roof with "instance" *thatched*). However, real-life applications are based on verbs (e.g., hiring, buying, selling, checking-in, boarding, etc.). To form the basis of a truly useful new environment, the objects that should be created and reused are therefore the verbs, or action (processing), as opposed to the nouns (data).

Further problems that arise in selecting a noun-based starting point are aggravated by inappropriate language goals. Currently, languages such as C++, Objective C (both derivatives of C), and Smalltalk are being marketed as the way forward in the implementation of object-oriented end-user applications. Technically, a strong case could be made for C being a 2.5GL. For example, memory allocation still remains in the hands of the programmer. The problem with a 2.5 or 3GL is that their vocabulary is fixed. Language structures in 5GL software must evolve and embrace new technologies and concepts. No matter how much application logic is written with third-generation languages such as C, the process does not enhance or expand the vocabulary of the language itself.

Purists may argue that using these languages allows developers to create objects and use them elsewhere. However, using them requires integrating their logic, again employing a fixed-vocabulary language. This approach would be like creating new native-language words (e.g., in English, French, German, etc.), then using a fixed vocabulary in, say, Latin to encapsulate a higher concept, and then requiring the use of Latin again to integrate them into sentences.

Like any spoken language that became fixed, earlier generation computer languages provide no foundations upon which to evolve new languages that can cope with change. Once a language becomes fixed, it ultimately ceases to be an appropriate form of expressing new ideas and communicating problems and solu-

continued on page 8

Object Design, Inc

Object Design, Inc. introduced Release 2.0 of ObjectStore, the object-oriented database management system software for networked UNIX workstations and high-end personal computers. It includes greater support for ObjectStore applications running in heterogeneous networks with multivendor server and client computers, support for CD-ROM optical media, and enhanced security features. Pricing for development licenses is on a per-seat basis, ranging from $1,495 to $6,000. Runtime license pricing is also available.

Object Design, Inc., One New England Executive Park, Burlington, MA 01803, 617.270.9797; fax: 617.270.3509.

Management Information Technology Inc

Management Information Technology Inc. (MITI) has developed SQL*C++, a software tool allowing developers to build object-oriented interfaces to relational databases without needing to change the company's legacy systems and applications. Price varies by platform starting at $500 for versions running under MS-DOS. Prices for UNIX platforms begin at $1,800.

Management Information Technology, Inc., 2895 Temple Ave., Long Beach, CA 90806, 310.424.4399; Fax: 310.424.9385.

Softeam

Softeam announced the U.S. introduction of OBJECTEERING and CLASS RELATION. OBJECTEERING is a powerful CASE product that automatically generates C++ applications from CLASS RELATION's object-oriented model. CLASS RELATION is a methodology that provides a single object-oriented model to encompass the entire software development life cycle from specification to coding. Both operate under UNIX on Sun, Hewlett-Packard, and DEC platforms. The product is $9,500 for the first copy with discounts available for multiple licenses.

Softeam, 1 Kendell Sq., Ste. 2200, Cambridge, MA 02139, 617.621.7091; fax: 617.577.1209.

Emergent Behavior

Emergent Behavior announced two new versions of their C++ framework solving problems using Genetic Algorithms: MicroGA 1.0 for Microsoft Windows and MicroGA Version 1.1 for Macintosh. MicroGA includes full source code to the library, three sample programs, and the Galapagos code generator, which allows users to create complete applications with MicroGA without writing any C++ code.

Emergent Behavior, 635 Wellsbury Way, Palo Alto, CA 94306 415.494.6763; e-mail: emergent@aol.com.

Computer Associates

CA-Common View class library extends Computer Associates' commitment to industry GUI standards including CUA 91. CA-Common View 3.1 integrates with CA-C++ 3.0 to enable efficient development of Presentation Manager 2.0 applications. A simultaneous release of CA-Common View 3.1 for Windows 3.1 integrates with Microsoft C/C++ 7.0 and Borland C++ 3.1. CA has adapted CA-C++ 3.0 for OS/2 2.0 keywords and switches.

Computer Associates International, Inc., One Computer Associates Plaza, Islandia, NY 11788-7000, 516.342.5224; Fax: 516.342.8329.

Suite Software

Suite Software announced that its Distributed Object Management Environment, SuiteDOME, is now available on the HP9000 Series 700 and Series 800 platforms. SuiteDOME Release 2.1. is both a distributed operating environment and development tool set for building interoperable distributed applications. SuiteDOME prices for the HP9000 Series 700 and 800 range from $750 for entry-level models to $12,000 for the high-end servers. Discounts are structured on a volume basis.

SuiteSoftware, 777 Alvarado Rd., Ste. 308, La Mesa, CA 91941, 619.698.7550; fax: 619.698.7567.

Versant

Versant Object Technology Corp. announced Release 2 of the Versant C++ Application ToolSet. Major enhancements include the addition of two new products, VERSANT Report, a report generator, and VERSANT Object SQL, which allows application developers to use queries based on industry-standard SQL to access the VERSANT ODBMS. Release 2 also includes enhancements to VERSANT Screen, a graphical application development tool.

Versant Object Technology, 4500 Bohannon Dr., Menlo Park, CA 94025, 415.329.7500; fax: 415.325.2380.

ObjecTime

ObjecTime 4.0, representing major enhancements to an object- oriented CASE tool for distributed, event-driven systems, was introduced by ObjecTime Limited. ObjecTime supports an advanced methodology for the analysis and design of distributed, event- driven systems known as Real-Time Object-Oriented Modelling (ROOM). The high-level object-oriented concepts are programming language independent.

ObjecTime Limited, 340 March Rd., Ste. 200, Kanata, Ontario, Canada K2K 2E4, 800.567.TIME, 613.591.3400; fax: 613.591.3784.

Product Announcements

Product Announcements is a service to our readers. It is neither a recommendation nor an endorsement of any product discussed.

Berkeley Productivity Group

Berkeley Productivity Group is shipping The Smalltalk Interface to Objective-C, a set of Objective-C and Smalltalk classes that make it possible to send messages from a Smalltalk process to an Objective-C DLL. With the interface, both Objective-C and Smalltalk can be used together, or portable applications engines can be written in Objective-C, moved to the platform of choice, and integrated with a new interface developed in Smalltalk.

Berkeley Productivity Group, 35032 Maidstone Ct., Newark, CA 94560, 510.795.6086; Fax: 510.795.8077

NovoTech, Inc

NovoTech, Inc. announced Objective-C v1.0 for PenPoint, providing the extensions necessary to use the PenPoint Class Manager without maintaining method tables and class registration functions. NovoTech is also providing IC Pak 101, the Objective-C foundation library for PenPoint. The Objective-C compiler is $150, IC Pak 101 is $80, and the Objective-C compiler/IC Pak 101 bundle is $200.

NovoTech, Inc., 88 Doolittle Dr., Bethany, CT 06524, 203.393.3729; 203.393.3730 (fax)

Component Software Corporation

Component Software Corporation announced the availability of COMPONENT WORKSHOP V1.0 for the Macintosh, a dynamic C++ application development environment. COMPONENT WORKSHOP includes an application framework, flexible set of integrated tools, full suite of C++ class libraries, and basic set of components. The Extruder, also included, frees the application from the dynamic environment, highly optimizing it and ensuring that applications are small and fast. The suggested retail price is $2,495.

Component Software Corporation, 420 Bedford St., Lexington, MA 02173, 617.862.9700; Fax: 617.862.7749.

Integrated Development Corp

Integrated Development Corp. announced the release of LibTools, a set of programmer's tools for creating, managing, and exploring libraries of C, C++, Assembler, Xbase, and all other Intel-, Microsoft-, and Borland-compatible object modules. LibTools works with object modules created with almost every compiler including C, C++, Assembler, Xbase, and others.

Integrated Development Corp., 190 Main St., P.O. Box 592, Hampstead, NH 03841; 800.333.3429, 603.329.5522; fax: 603.329.4842.

Lucid, Inc

Lucid, Inc., announced XLT, a set of productivity tools for its Lucid Common Lisp programming environment. Version 1.0 of XLT is shipping on the Sun SPARC platform and compatible hardware, HP Series 7600, IBM RiscSystem 6000, and DECsystem. XLT provides an X-windows system-based interface to data inspectors, program and data analyzers, a stepper, debugger, and other tools including an interface to GNU Emacs. XLT lists at $1,800. Site licenses are also available. Customer support starts at $480/yr. including future updates of the product and hotline support.

Lucid, Inc., 707 Laurel St., Menlo Park, CA 94025, 415.329.8400; fax: 415.329.8480.

Vermont Creative Software

Vermont Creative Software's cross-platform application-development tool now supports every major MS/PC-DOS extender. Vermont Views Plus allows developers and programmers to "draw" the user interface without writing, testing, and debugging thousands of lines of computer code. Vermont Views Plus costs $795, yet contains all the tools of the standard edition of Vermont Views plus supplemental add-ons that formerly sold at a combined cost of almost $1,600 including Vermont Views MemEx, which supports all major DOS extenders including the royalty-free extenders bundled free with compilers such as Borland C++ 3.1 and Intel 386/486 C.

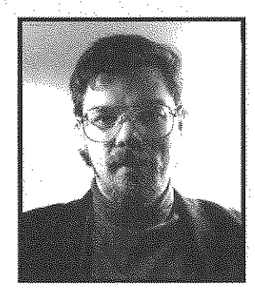
Vermont Creative Software, Pinnacle Meadows, Richford, VT 05476, 802.848.7731, 800.242.1114; Fax: 802.848.3502.

MADA

MADA announced that ACIUS' OBJECT MASTER 1.0 is now available to US and Canadian MADA members. OBJECT MASTER is an integrated Macintosh source-code editor that works with all Macintosh software development environments, with editing and navigation features designed specifically for O-O programming. The MADA member price for OBJECT MASTER is $320 (retail price $395).

MADA, 10062 Miller Ave., Ste. 202-B, Cupertino, CA 95014, 408.253.2765; Fax: 408.253.2767.

Infrastructure for a new economics of software



Sergiu S. Simmel

In an industry embracing object-based technologies at a rapidly increasing rate, one might have predicted that the software trade would correspondingly migrate toward a world in which small-to-medium software components and aggregates of such become a dominant form. Instead, we are experiencing very little if any change from tradition. The products traded continue to be the very large, monolithic software conglomerates we call applications, platforms, and environments. Why? What happened to the object-based technologies' promise of encapsulation, modularity, and reuse? Why haven't they broken and restructured the shape, size, and economics of software products themselves? Why has the popularity of object-based technical principles, languages, and architectures not yielded a similar growth in the market for software components?

Brad Cox attributes this phenomenon to the fact that "we have never developed a commercially robust way of buying and selling easily copied intangible goods like electronic data and software."[1] And he is right. We still sell software like potatoes. We forget that while potato growers have a good handle on how many potatoes there will be on the market (it takes two years for a potato to germinate and yield another potato), software "growers" don't have a clue: It only takes a few computer instructions and a bit more disk space to clone a piece of software.

As Miller,[2] Cox,[1] and a few others[3,4] have pointed out, the key to the solution of this economic problem is to decouple revenue collection from software distribution, i.e., turn the ability to easily clone software from a liability into an asset. This amounts to distributing the software itself either freely or for a very low price (merely to cover media, manufacturing, and shipping costs) and charging the customer for its use only.

Charging for use (sometimes referred to as pay-per-use, metered use, or meterware) is a new concept in the software industry, but not a very new one in general. Other industries have employed it for a long time. Your gas company installs a meter in your basement, reads the meter every couple of months, and charges you only for the gas you actually used. Your telephone company does the same with your long distance calls. The City Hall collects parking fees from meters (both as voluntary charges and as fines). Even the US Postal Office does a similar thing when it fills up your postage meter, except it charges you ahead of time—more like your local gas station!

In many industries, pay-per-use has been established as a major and often dominant method. However, even in these industries pay-per-use was not always the preferred method. Still, one can see that the adoption of pay-per-use has been a sign of maturity and economic realism.[4] We believe that the same evolution is now taking place in the software industry.

Cox refers to this approach of distributing software distribution and collecting revenues as "revolutionary."[1] Throughout the rest of this article, I will try to suggest that this approach can actually be made as evolutionary as you'd like by complementing it with the more familiar "floating licensing."

The mechanism used to implement this approach is critical to its success. In practice, the mechanism required is very similar to that used to manage persistent data. It is another facet of the general management of visibility of data and services. It is vital that the mechanism be constructed so it leaves policy entirely in the hands of software manufacturers. This is necessary to ensure that business and market considerations rather than technical impositions are the main factors determining policies. Throughout, we will be using examples drawn from the Kala Persistent Data Server—a software component product from Penobscot Development Corporation—which has provided an implementation of these "new" concepts for several years now.

KALA AND METERED SOFTWARE

Kala is a persistent data server. It is a software subassembly with two areas of functionality:

1. managing the storage and retrieval of arbitrary data in a high-performance, compact, recoverable, shared, secure, distributed, and robust manner
2. managing the visibility of data for flexible and inexpensive support of arbitrary transaction, access control, versioning, and configuration models.

The basis for metered software is the observation that objects are trivially able to monitor their own use[1] so long as the monitors can be safely and securely stored persistently. Usage monitors are nothing but data, and Kala's specialty is precisely the secure and safe storage of data. This makes it the natural locale in the architecture to deal with these issues. The only question left is how we can associate the usage monitor and the datum.

The Kala facility supporting the pay-per-user approach is called SoftMeter. The SoftMeter works like a postage meter except it parcels out abstract *meter units* instead of postage. Applications can debit the meter (using Kala's API function Debit-

Meter()) down to empty. The putmeter utility program refills the meter using a key provided by the manufacturer or its resellers. The same facility is also expressed through Kala's API by the RefillMeter() and MeterBalance() primitives.

DebitMeter() debits a given number of meter units from a meter counter for the current application's vendor. DebitMeter() succeeds silently if there are at least the required number of units left in the SoftMeter. If there aren't enough meter units left to satisfy the debit then Kala raises an exception.

In the spirit of flexibility, Kala allows you to replace its own (default) persistent data access method with any arbitrary code. You can do this on a per-data-item basis.[5,6] For example, you can supply a routine that loads the datum from the persistent store using Kala's Load() primitive and then decompresses it (or decodes it) using your own decompression (or decoding) algorithm. Because the code you supply can be arbitrary, it may call any Kala primitive. In particular, it can contain a call to DebitMeter().

Vendor ids are assigned by Kala's manufacturer, which guarantees their uniqueness and establishes a channel through which vendors can obtain revenue. Meter units are sold by the Kala manufacturer or resellers. DebitMeter() uses the vendor identifier to record in the Kala Store the running total of debits broken down by vendor. Periodically this information is transmitted to the manufacturer or its authorized reseller, who pays to the corresponding vendors an appropriate share of the proceeds of meter unit sales. While this process is likely to be partly manual at first, it is fully mechanizable to the point of requiring no human intervention.

The scheme is flexible enough to impose few if any constraints on the business model of an actual manufacturer–reseller–vendor–customer relationship chain. In particular, the point of revenue collection and redistribution can be located either with the manufacturer, reseller, or any of the vendors in between.

Using DebitMeter() in conjunction with the non-default access methods and other visibility control primitives, you can implement a very simple data access and creation metering policy, a very sophisticated one, or something in between. For instance, you can choose to meter the access only to certain data; charge a flat fee for any use of the application (regardless of how much data is actually accessed); meter only the initial access to a cluster of data but not to individual components; vary the amount debited depending on the data, etc.

Kala data can be anything that can be represented as bits and pointers. In particular, it can be executable code. When programs or dynamically linked libraries are stored in Kala, their use can be metered using the same general principle.

As explained above, Kala's SoftMeter is used to monitor the access to data or programs stored in the Kala Store. How about Kala's own services? When in metered mode, Kala will charge for each of three fundamental operations: attaching to a Kala Server, loading a Kala datum, and creating a new Kala datum. Kala uses the same DebitMeter() primitive internally, but now it doesn't charge on behalf of the datum's vendor, but on behalf of Kala's manufacturer. The vendor can have confidence in the meter system because Kala itself uses it for its own revenues.

Kala's ability to ensure the safety and security of the metering information is one of the keys to the success of this scheme. Not only can Kala's identifiers not be forged, but also Kala's access to its data is as secure as the operating environment. Moreover, Kala's dynamically managed Persistent Store makes it impossible for someone to even pinpoint where the metering information is stored. Kala rearranges the Store dynamically to increase performance and collect garbage and, in the process, moves data around, making clever hacks difficult. Even hacking application code to branch around the DebitMeter() calls is detectable when the application has been well integrated with Kala.

PREDICTABLE USAGE AND LICENSES

Basing a major portion of software trade on mechanisms like Kala's SoftMeter will require substantial changes in the design, implementation, and packaging of the software being traded, as well as in its quality. Such changes will not happen overnight. Thus, in addition to metering, support for conventional licensing is still essential. It turns out that a proper implementation of the floating license preserves the principle of separation between distribution and revenue collection mentioned above.

Licenses do not grant ownership to a piece of software. They only grant the right to use it. A Kala license opens a path between a Kala Store and the application. A License makes the Kala Store visible to the application.

An application (Kala client) acquires a license by calling the AcquireLicense() API function . Licenses can be general (any application can acquire the license) or vendor specific (only a certain application or an application from a certain vendor can acquire it). The function silently returns true if successful, false if not. The application decides what to do if acquisition fails. Acquired licenses may be surrendered by a call to ReleaseLicense(), and any remaining acquired licenses are automatically released when the application detaches from Kala.

There is an interplay between the metering and licensing mechanisms. On the one hand, they are independent of each other. An application can decide to run off the SoftMeter or use a license. On the other hand, if AcquireLicense() actually fails but the application decides to continue, its Kala-related activity will automatically be charged against the SoftMeter resources the Kala Store may have, if any, for this application. If there are no meter units left, Kala will raise an exception. An application can use the same policy for its own licenses, silently segueing to metering if it finds no license installed.

This dual mechanism is essential to the evolutionary approach Kala introduces to software distribution and revenue collection. Let's look at a typical example:

Suppose that you are the manager of a medium-size software development group, say, 30 people. You have 25 software engineers and 5 software writers. Your platform is a network of UNIX workstations. You are faced with the purchase of an authoring system to be the writers' main tool, which also must be usable by the engineers. Typically, you'll be presented with a floating license-based product. You know that your writers will use the system intensively every day. You also know that your engineers will rarely use it, but once a month they will all want to use it simul-

1. **Communication layer** (sends, receives, routes, and queues messages).
2. **Interface layer** (provides translation between communication and computation).
3. **Computation layers** (applications, database managers, objects).

In general, as the coupling between applications becomes weaker the middleware API level of abstraction in the interface layer should increase (e.g., a lower middleware API in the interface layer might be adequate for tightly coupled applications while a middle middleware object API will be necessary for many medium-coupled application frameworks). This provides modularity and facilitates the maintenance and enhancement of communication and computation components without disrupting the entire system.

Some of the major middleware challenges for system architects and developers are:

- What combination of middleware tools should be employed in building the new applications?
- Which level of middleware API should be used in specific applications?
- What standardizations are required in the middleware area to ensure future interoperability, portability, and maintainability of distributed data and application environment ?
- How can legacy environments be integrated or migrated into the new applications?
- How should the system design, development, maintenance, and management process be changed to cope with the new distributed heterogeneous applications?

CONCLUSION

In the next few years, there will be a rapid growth in the implementation of distributed applications using cooperative processing. The new middleware products will make it possible to extend these applications across heterogeneous and legacy environments. This will provide an opportunity for a cost-effective reengineering of current business processes. The challenge for system developers and business planners is to develop a strategy for an orderly transition to the new processes. This will require a detailed knowledge of the new middleware products including their capabilities and weaknesses. The development of standards enabling portability and interoperability will be also be crucial. The integration of object-oriented technology and communications middleware will be a key factor in building future distributed object environments. ≡≡

Reference

1. King, S. Message-delivery APIs: the message is the medium, DATA COMMUNICATIONS 21(6): 85–95, 1992.

Robert Marcus holds a B.S. and PhD in Mathematics and taught Mathematics and Computer Science at the City University of New York. He previously worked at the Hewlett-Packard Knowledge Systems Laboratory. He is currently a Coordinator for Object-Oriented Technology at Boeing's Research and Technology Center. In 1991, he initiated the CFOOT end-users group.

Industry Briefs

An independent Canadian software company, **ObjecTime Limited**, has been established as a spin-off from **Bell-Northern Research** (BNR), the research arm of **Northern Telecom**. Northern Telecom has granted ObjecTime Limited world wide license rights to the software design tool technology originated at BNR's lab in Ottawa, Canada.

Grady Booch and **Benjamin/Cummings Publishing Co.** jointly announced they will develop a new series of books devoted to important contributions in object-oriented technology called the Benjamin/Cummings Series in Object- Oriented Software Engineering. It will be launched next year with two books by Booch. Both Booch and Benjamin/Cummings said they will accept proposals from prospective authors via e-mail,and US mail.

The **Object Management Group** (OMG) announced that **British Telecommunications plc** has joined enduser participation in the consortium. In addition, **Lotus Development Corporation** and **United Technologies** have joined as corporate members.

Guidance Technologies, Inc. together with **Fujitsu Systems Business of Canada** (FSBC), a subsidiary of **Fujitsu Ltd.**, Japan, announced they have signed an agreement for FSBC to become the exclusive Canadian distributor of Choreographer. FSBC will assume responsibilities for both the sales and support of Choreographer in Canada.

NCR Corporation and **Objectivity Inc.** announced the availability of the Objectivity/DB object database management system on NCR's System 3000 computers, and a partnership to jointly market both products.

Objectivity Inc. announced a partners program for vendors of object information technology designed to provide marketing and technology partnerships that provide application development tools to increase software development productivity for object database applications. The first participants are **Digital Equipment Corporation** (DEC), **Hewlett-Packard, NCR, CenterLine Software, ProtoSoft, Inc., Associative Design Technology, Persistence, Interactive Development Environments** and **ParcPlace**.

SunPro announced a technology development and licensing agreement with **Rogue Wave Software, Inc.** The agreement centers on Rogue Wave's Tools. h++ class library product, which SunPro intends to distribute at a future date as an add-on to its SPARCworks Professional C++ development environment.

dors and standards groups to work on the requirements of end-user corporations.* In this time of rapidly changing technology, there has arisen a proliferation of standards groups, multivendor consortia, proprietary products, and systems integrators. It is essential that end users become well informed and proactive in comprehending and evaluating new developments and trends. Our goal can be stated as: "Technology + Business Case + Migration Plan = Success." As part of our work, the CFOOT group has raised a number of initial requirements related to middleware. These are listed below for your information:

- Requirements for object-oriented technology vendors
  1. Consider your products as components of larger and/or legacy systems.
  2. Focus on areas where your product can add value.
  3. Avoid unnecessary proprietary features and interfaces.
  4. Interoperate with other products and components.
  5. Plan to interface with multiple legacy platforms, systems, processes, and organizations.
  6. Supply tools for supporting total lifecycle project management, methodologies, and metrics.
  7. Provide robust class libraries to encapsulate non-object-oriented systems.
  8. Provide application- and interface-generating tools built on top of these databases, libraries, and encapsulations.
  9. Provide methodologies, tools, and training for object-oriented system analysis, design, implementation, management, and maintenance of large software projects. In particular, estimation/planning and code/test generation tools.
  10. Supply specification, code-generation, and test-generation tools.
- Requirements for object-oriented technology standards and multivendor groups
  1. Standards and tools for interoperability among all encapsulations, databases, languages, and libraries.
  2. Standard interfaces for portability across databases.
  3. An information system framework (environment) that provides services for individual objects such as communication, translation, resource management, and ownership.
  4. Repository specifications to aid in the configuration control, integration, and management of the libraries, tools, and encapsulations.
  5. Object model and terminology standards.

THE MIDDLEWARE CHALLENGE TO BUSINESS PROCESSES

The expanding capabilities of middleware products provide a tremendous opportunity for business process reengineering. It is important that these changes take place in a controlled, planned fashion to avoid disrupting the corporate environment. Planners must begin considering the implications of the new technology now. Enterprise data and legacy applications will, theoretically, be accessible from many distributed desktops. It will be possible to build integrated automated workflow processes within and across functional boundaries. Applications like concurrent product definition, sophisticated document flow management, and large-scale computer integrated manufacturing will be far easier to develop.

There are many challenges that must be addressed to realize the full potential of middleware for business processes. Some of these challenges are:

- Given that applications will soon be able to communicate and retrieve data throughout an enterprise in a timely fashion, how should the business processes be changed to reduce cost/flowtime and increase product quality?
- What types of distributed enterprise-level applications must be developed using the new middleware capabilities to support the new business processes?
- How can legacy processes be integrated or migrated into the new processes?
- What cultural and institutional changes are necessary to introduce the new processes?

THE MIDDLEWARE CHALLENGE TO SYSTEM ARCHITECTS

The design and implementation of the new middleware-based distributed applications will be very difficult due to their heterogeneous environments and the wide range of alternative approaches that must be considered. In general, the coupling level between the system components is a crucial factor for decision makers. One possible classification of levels might be:

1. Tightly coupled
   - Single applications with distributed services.
   - Client–server RPCs.
   - Direct communication between client and server.
   - Fine-grained objects encapsulating procedures.
2. Medium coupling
   - Application frameworks with multiple tools.
   - Peer–to–peer, broadcast, notification, shared services.
   - Communication in a homogeneous domain.
   - Medium-grained objects encapsulating tools.
3. Loosely coupled
   - Multiple independent applications including legacy systems.
   - Peer–to–peer messaging with queuing and reliability.
   - Communication across multiple heterogeneous domains.
   - Large-grained objects encapsulating applications.

From a middleware viewpoint, there are three layers that must be developed for new systems:

taneously to write their status reports to management. The question is: how many floating licenses should you buy?

If you buy 5, then either no engineer will be able to use it or they will constantly fight over licenses with the writers. What a waste of time, energy, and perhaps even people! If you buy 30, then everybody will be happy, except for your CFO: you will end up with 25 licenses sitting around unused for most of the time. What a waste of money! If you buy any number between 5 and 30, you'll be worse off because you'll still waste purchasing money and have to force people to work at odd hours to use their licenses.

With a Kala-based authoring system, you don't have to worry about any of the above. You buy 5 licenses to satisfy the predictable, steady use by the writers. You also buy some amount of meter units, and have the application run off the meter any time more than 5 people try to use it. If you run out of meter, you call your vendor with your credit card and buy more. Problem solved: save both headaches and money at the same time with a simple but flexible approach.

Licensing and metering are not mutually exclusive. Instead of replacing each other, they complement each other. The licensing mechanism is the best fit for predictable, relatively constant use. The metering mechanism is the best fit for intermittent bouts of heavy use. Since both levels of activity are common in the actual practice of using a shared application, a combination of metered and licensed software offers the best of both worlds.

THE SUPPLIER'S SIDE

Over the past few years, the food chain between the infrastructure software developer and the consumer has been continuously lengthening, and this will increase in the future. Platform software vendors often sell to middleware vendors who often sell to application vendors who often sell to integrators who finally sell to the end user. Licensing and metering must trace these often complex paths and allow the proper disbursements of revenue.

A licensing and metering mechanism must be flexible enough to allow negotiations and business deals to proceed smoothly, unencumbered by technical limitations. For example, the mechanism must be totally recursive: each consumer should be able to turn around and become a supplier and, in the process, use the same mechanism to satisfy its needs without jeopardizing the negotiated rights of its own suppliers.

Kala accomplishes this by offering access to the entire mechanism through its administrative API. You can install licenses in a Kala Store using the API InstallLicense() and ShowKalaLicenseStat() functions. Licenses hold information about the vendor (for vendor-specific licenses), the duration of license validity (as short as a day and as long as infinity, to provide perpetual licensing), and the number of seats (licenses).

Installing a license requires a key the user obtains from its application vendor. Application vendors get the newlic program that generates such keys when they obtain their vendor id from the manufacturer. The newlic can only generate keys for that vendor, not keys for licenses for vendors with other ids. Thereafter, the Kala manufacturer need not be involved in the license granting, sales, or administration of the application vendor. An alternative is to have the manufacturer dispense licenses on behalf of the vendor, at lower cost due to volume.

As with the SoftMeter facility, Kala Stores hold licensing information persistently in a totally safe and secure manner. The use of a "cookie" (an identifier tag produced at the Kala Store site by a supplied utility) in fabricating the license key guarantees a one–to–one mapping between licenses and Kala Stores.

The supplier using Kala licensing and metering can use a conventional distribution system through retail stores, dedicated sales staff, or mail order. However, the Kala system opens another possible distribution model that may be appropriate for many software developers. For example, suppose you are a small garage shop that has produced a multiplayer interactive game program that stores game history and player state on Kala. You can place the game itself on bulletin boards for free and let users try it out on the (small) SoftMeter balance that comes with every Kala system without charge. When play eventually exhausts the meter, your players will contact the manufacturer or reseller for as many licenses (or meter refills) as they need. You, in turn, get a regular check from the manufacturer or reseller without sales force, marketing costs, or hassles. The pricing for each vendor is subject to negotiation, and may be structured in any fashion deemed necessary to satisfy business needs.

This model is obviously very much like *shareware*. It has the shareware virtues of low distribution costs (from the developer's point of view) as well as low trial cost and, hence, low risk (from the customer's point of view). With Kala, unlike shareware, the developer's revenue is not determined by the conscientiousness and good will of the customer but rather the license and meter policies and prices chosen by the developer.

CONCLUSION

As the software industry matures, its dominant methods for software distribution and revenue collection are expected to migrate toward a combination between floating licensing and metered use. To allow the industry to settle on the methods most likely to provide long-term revenue streams and discourage massive fraud and inequity, the proper infrastructure must be put in place.

Metering and licensing are complementary, not mutually exclusive. They should be used in conjunction to cover both predictable and random use, both constant and intermittent usage.

Metering and licensing are specific ways of controlling the visibility of data and services based on paid-for rights. To ensure the safety and security of metering and licensing information, the metering and licensing mechanism must be located where persistent data is stored and guarded—in the persistent data store. At this low level, a persistent data server such as Kala can ensure the protection and inviolability of licensing and metering information, thus guarding the economic interests of all parties involved in the software trade: manufacturer, resellers, vendors, and customer. ≡≡

References and suggested reading

1. Cox, B. What if there is a silver bullet, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, 5[3]: 8–9, 76, 1992.

continued on page 12

tions. For example, suppose English were fixed on a finite set of primitive vocabulary, prior to the invention of airplanes, and a "user" wants to express the concept of "airline check-in procedure." This would have to be expressed in fixed "building blocks" such as "a mammal standing upright on two legs with two arms walks over to the square object made of wood and makes sounds to another mammal standing upright on two legs . . . ." As you can see, this rigidity in a vocabulary introduces inconsistency in expression, incompleteness, inaccuracies, and conflict. However, software implementations still suffer from these problems. Therefore, computer languages and their evolution should be treated no differently than the necessity that drove the evolution of spoken languages.

With the right focus, 5GLs can now be built via object-oriented procedures. With the development of verb objects, reusable building blocks can be created that express business processes. These, in turn, can be rapidly integrated into large systems, which can quickly embrace not just existing "legacy" applications but also the ever-increasing base of new technologies and enhanced requirements. As usual, the $64,000 question is, "Can this be true and, if it is, can it be done today?"

EXISTING SOLUTIONS

The answer to both these questions is an emphatic "Yes." At least one development exists today, called MacroScope, that has all the 5GL capabilities briefly outlined here plus many more. Its design goal was to provide application developers with the type of accelerated implementations achieved by hardware evolution. The architecture that evolved in MacroScope started with the creation of detailed layers of computer-science building blocks. These then evolved into a language, first of high-level "technology verbs," then into high-level "business verbs." These verbs correspond to some extent with methods in conventional object-oriented parlance, but their different design provenance makes them much more powerful. The infinitely flexible logic of an application is achieved by the sending and receiving of messages, much as one person might call another on the phone.

In fact, this parallel with the way people work is central to the extended object-oriented paradigm in MacroScope. For example, objects can be built that know how to buy, sell, evaluate a portfolio value, etc. These can now be used in, say, a dealer trading system, in treasury cash management, for investment analysis, etc. A developer simply invokes the "sell" object/verb: what it then does (its processing), in terms of database access, communications, and logic (e.g., technology objects being invoked), is transparent to the developer.

This layered, evolutionary approach to software in fact provides a programming revolution; namely, programming languages with extensible vocabularies. Due to object-oriented capabilities, an implementation language with this "organic" structure can grow into even higher components. This is achieved by the creation of development vocabularies that embody specific industry or corporate processes. What this development revolution provides is the ability for more people to create their own solutions, like the impact of the calculator on use of the slide-rule. The calculator allowed a very wide audience of users to work effectively with mathematics without needing to understand algorithmic detail. Users can now apply/employ high-level business concepts without needing to understand the detailed processing logic underneath the concept (e.g., a monthly payment based on principle, interest, duration of time, etc.)

THE CHALLENGE

A fifth-generation object-oriented language dramatically shifts traditional programming paradigms, so much so that a (natural) first reaction is incredulity. This skepticism is to be encouraged. This article started by highlighting the excessive claims that have always been made for software. In fact, it is difficult to make credible comparisons between a 5GL and traditional application development approaches. With a 5GL, complex solutions can be developed in man months of effort, and weeks of elapsed time. How does one make a believable comparison between applications that have been written over two to five years, by hundreds of programmers, and a solution that can "redevelop" that application in three months, with more functionality, and interfaces that require virtually no training or documentation and are capable of running on smaller, cheaper, more robust hardware?

A practical way of removing this incredulity is to have users "lay down a gauntlet" and have vendors "pick it up" and put their money and resources where their marketing muscle has been. This is certainly something that MacroScope has done many times in the years during which it was growing to maturity: many complex applications have been created in man months (even man days) that otherwise would have taken many man years to build. This is an appropriate way to combat skepticism. In fact, in an industry that has never been reticent in making outstanding claims for itself it could be an interesting approach. If the technology delivers on its claims, there should be no shortage of organizations willing to take it up. This could be really useful to the wide audience that must be extremely interested in knowing whether the future has really arrived for software development. Take this approach. Challenge us tool vendors to prove that our products can deliver more than traditional languages at a lower cost. Some of us can and will rise to the challenge.

In this era of rapid and accelerating change in all business climates, very large programming backlogs in all environments, aging legacy systems, and ever-rising software maintenance costs, developments can no longer be viewed in the brute-force context of "everyone program faster." What is needed is the next leap forward in technology: the equivalent that "Beam me up, Scotty" was to ground–to–air transportation. Perhaps an object gauntlet would help us know for how much longer we will have to use the people-mover, shuttle, and airlock. ≡≡

Bernadette G. Reiter is founder, President, and CEO of Objective: Inc., the manufacturers of the Macroscope fifth-generation development environment. Based in Boulder, CO, Objective: Inc. provides custom object-oriented software solutions to Fortune 500 companies within the US and abroad.

network protocols: TCP/IP, DECnet, Netbios
operating environments: VAX/VMS, Netware NLM

- **Peerlogic:** Pipes; 415.626.4545
  Peerlogic supports DOS, OS/2, UNIX, Netware, and MS-Windows. A mainframe version is under development. The product claims to provide dynamic and flexible communication across many heterogeneous network protocols. Peerlogic has just joined the OSF and plans to build a DCE services-based environment using their product as an alternative to the DCE/RPC. Clients include American express, DHL Airways, and Unum Life Insurance.

  network protocols: LU6.2, TCP/IP, Netbios
  operating environments: IMS

- **Suite Software:** SuiteTalk, SuiteDDM, and SuiteDOME; 619.698.7550
  Suite Software makes a messaging product called SuiteTalk, a distributed object management system named SuiteDOME, and a distributed heterogeneous database management product named SuiteDDM. All these products have just been released. In addition to the currently supported environments, ports to Netware, LU6.2, MS-Windows, and OS/2 are underway.
  The Suite Software product set seems to provide the most complete set of functionality of all the messaging-related products. SuiteDOME does not correspond to the OMG Object Request Broker specification but they have stated their intention to converge toward industry standards. Customers include American Airlines, NASA, and JPL.

  network protocols: TCP/IP, DECnet
  operating environments: VAX/VMS, UNIX

- **SUN:** ToolTalk
  A messaging facility, ToolTalk, will be bundled with OpenWindows. It is built on top of the TI-RPC product. ToolTalk has received early support from the CAD Framework Initiative and the CASE Interoperability Alliance and has been licensed by Silicon Graphics. It is not certain what other environments it will support, athough the TI-RPC should allow it to be very portable.

  network protocols: TCP/IP.
  operating environments: Solaris

- **Symbiotics:** Networks; 617.876.3635
  Symbiotics has a sophisticated product based on distributed agents communicating by messaging. It does not support as many platforms or protocols as the other vendors. The Networks product has an object-oriented foundation that may be an example of the future direction for some other messaging products.

  network protocols: TCP/IP
  operating environments: Unix

- **System Strategies:** Ezbridge Transact; 212.279.8400
  System Strategies is a subsidiary of the NYNEX phone company. Its product runs on CICS, OS/2, DOS, and VAX/VMS. The company has an agreement with Transarc to interface Ezbridge with Transarc's Encina transactional RPC. This could provide a gateway between OSF/DCE environments and asynchronous messaging capabilities. System Strategies is the largest of the non-hardware messaging vendors.

  network protocols: LU6.2, TCP/IP, DECnet.
  operating environments: IMS, VAX/VMS, Netware NLM

- ANSA (Advanced Networked Systems Architecture); dme@ansa.co.uk
  ANSA is a European Esprit project with similar goals to the OSF distributed computing environment. ANSAware implements their architecture. It has been used by NASA to build a wide-area distributed information system. ANSAware is based on an RPC foundation.

- BBN: Cronus
  Cronus is a distributed computing environment that provides asynchronous execution by means of an RPC.

- **Gradient Technologies Inc.**; 508.562.2882
  Gradient has built a version of the OSF/DCE RPC for PCs.

- **HyperDesk;** 508.366.5050
  Hyperdesk has built the first commercial implementation of the OMG's Object Request Broker. It provides an object layer over RPCs like Netwise or DCE. Hyperdesk supports asynchronous behavior in its product.

- **Netwise;** 303.442.8280
  Netwise offers an RPC supported by Sun and Novell. Netwise has been used in industrial applications and supports a wide range of network protocols and platforms including IBM mainframes. Netwise's products include RPC TOOL for MS-Windows, RPC EXEC for MVS/CICS, and DUET for multi-protocol client–server RPCs.

- **Open Software Foundation's Distributed Computing Environment** (DCE RPC)
  Based on the HP/Apollo Network Computing System, this RPC has the most multivendor support but has not yet been used in any large-scale commercial applications. (Candle Corp. has just announced a joint development agreement with OSF to port the DCE RPC to MVS.)

- **Transarc:** Encina; 412.338.4420
  Transarc has built a transactional RPC based on DCE RPC.

- **Wonderware:** NetDDE; 714.727.3200
  Wonderware NetDDE extends the functionality of MS-Windows DDE across network protocols including NetBios, DECnet, and TCP/IP.

CFOOT

Understanding management, technology, and products (like those listed above) is the reason behind the Corporate Facilitators of Object-Oriented Technology. CFOOT is a mailing-list based group with over 200 subscribers, formed to influence ven-

the RPC client–server type of communication may be adequate and could be simpler to program.

The efforts of many multivendor consortiums seem to be focused on this RPC client–server approach, e.g., the OSF/DCE and the OMG Object Request Broker.

At the same time, there are many commercial products appearing that provide synchronous messaging capabilities across heterogeneous networks and platforms. Unfortunately, most of these products have proprietary interfaces. Some of the strongest supporters of the OMG/ORB approach are also working on independent messaging products (e.g., Sun's ToolTalk and HP's Broadcast Message Server). It is unclear how these products will interoperate with OSF/DCE, OMG/ORB, or each other.

The APIs for the higher levels of middleware have to encapsulate the capabilities of the lower levels to enable portability and interoperability. This is much more difficult than it sounds. Several probelms already being addressed by the OMG are:

1. ORB Object-Level Specifications are not specific enough to guarantee interoperability or portability between different environments.
2. ORB Object-Level Specifications are not adequate to encapsulate peer–to–peer communication (e.g., message queuing, broadcasting, notification).
3. There may be a need to build an upper middleware layer on top of ORBs.

MESSAGING PRODUCTS

Almost all vendors of middleware application intercommunication products support MS/DOS, MS-Windows, and OS/2, and some support UNIX. In general, the products provide message communication, management, and queuing facilities with relatively simple APIs. This corresponds to lower middleware. An important next step will be building middle middleware frameworks on top of these messaging capabilities (e.g., Suite Software and HyperDesk below).

An excellent detailed survey of some of these products can be found in an article by Steven King in DATA COMMUNICATIONS.[1] To assist readers wanting to explore these products further, a list of some current offerings follows including information about some of the network protocols and operating environments currently supported.

- **Covia Technologies:** Communications Integrator; 708.518.4000

  Covia Technologies is the system integrator for a very large airline reservation system used by United and other airlines. Covia was part of the initial development effort for LU6.2. It has been ported to Tandem, Stratus, Unisys, MS-Windows, VAX/VMS, MS-DOS, and OS/2. UNIX and Novell support are under development. Covia's product is designed around OSI system management but can link to Netview and Netmaster.

  network protocols: LU6.2, Netbios
  operating environments: IMS

- **Creative System Interface:** Application to Application Interface (AAI); 508.872.0965

  Creative Systems began with CICS, MVS, VTAM, DOS, and OS/2 functionality. Since then, AS/400 Windows and AIX versions have been released. Upcoming ports to DEC, Netware, and other UNIX platforms have been announced. Clients include Dun & Bradstreet. AAI has been licensed by Micro-Focus for its COBOL product.

  network protocols: LU6.2, Named Pipes
  operating environments: IMS

- **DEC:** DECmessageQ

  DECmessageQ began as a tool for linking manufacturing applications but is being positioned as an enterprise integration tool. It currently supports VAX/VMS, Ultrix, OS/2, DOS, and Macintosh. Linking it to LU6.2 and TCP/IP requires some additional development.

  network protocols: LU6.2, TCP/IP, DECnet
  operating environments: VAX/VMS, Mac

- **Hewlett-Packard:** Broadcast Message Service (BMS)

  BMS is used by SoftBench. There is a multivendor group called CASE Communique that is encouraging the use of BMS as a CASE tool communication standard.

  network protocols: TCP/IP
  operating environments: HP-UX, Sun

- **Horizon Strategies:** Message Express; 617.444.7575

  Message Express is known for its flexibility and functionality. It can deliver messages reliably even when networks, applications, or platforms are temporarily unavailable. It was initially developed for IBM mainframe and DEC VAX machines but has since ported to smaller platforms. Message Express can serve as a backend to the Sybase Open Server. Customers include Scott Paper, Munich Reinsurance, and the New York Power Authority.

  network protocols: LU6.2, Named Pipes, Netbios
  operating environments: IMS, VAX/VMS, Netware NLM

- **IBM:** DAE or Datarade

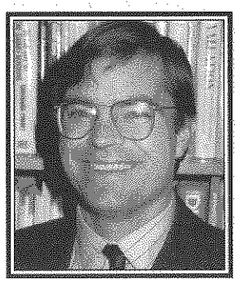
  The IBM DAE was originally developed for manufacturing system integration but is being repositioned as an enterprise integration system. Datarade is a product developed by IBM's financial services group. It does not currently support mainframe platforms.

  network protocols: LU6.2, TCP/IP, Named Pipes
  operating environments: AIX, OS/2

- **Momentum Software:** X-IPC; 201.871.0077

  Momentum Software supports semaphores and virtual shared memory as well as messaging in a single API. The initial releases of X-IPC were for UNIX and OS/2. Since then VMS, MS-Windows, and DOS have been added. Coming ports include Novell, MVS, and Macintosh. Customers include American Airlines, FMC, and Canadian Pacific.

You can have your objects and be relational too!



Christopher Keene

You may be one of those fence sitters who concedes the benefits of object orientation but complains that, because of the investment your company has made in relational databases, "there's no way to get there from here." This article describes how to make a smooth transition to object technology by building object-oriented business applications that store and access data in relational databases.

This article will cover three points: (1) Companies need a transition strategy that enables them to build new object-oriented applications on top of existing relational databases; (2) Connecting objects to relational databases is a straightforward but very time-consuming job; (3) Tools are available that automate this task, enabling companies to build applications that treat relational databases as if they "understood" objects.

COMPANIES NEED A TRANSITION STRATEGY

Every significant new technology brings with it two opposing camps of prophets: those who claim that the only way to get there is to abandon everything you have and take up the new standard, and those who claim that the risks are too great and it's better just to stay put. Not surprisingly, object technology has spawned its own set of fanatic believers and stern detractors.

Nowhere is this conflict as apparent as in the database arena. On the one side, object database gurus label relational databases as dinosaurs ready for the relic heap. On the other side, relational database vendors deride object databases as toys, not worthy of serious consideration.

Reality falls somewhere between these positions. Relational databases, whatever their other faults, are extremely flexible and can serve quite well as object repositories. With careful planning, companies can make the transition to object-oriented development by starting with existing relational tables. Over time, they can use this approach to create object-oriented applications that leverage existing data and work side by side with legacy systems.

Relational Data Has High Inertia

Although relational databases have been around for more than 20 years, hierarchical and network databases still account for more than 50% of all corporate data. The number one rule of database management is that data has high inertia. Thus, any new object application that needs corporate data is going to have to find a way to access the data "where it lives."

While object databases may eventually take over the database world, the revolution will be a long time coming. What companies need today is a viable transition strategy that will allow them to begin using object-oriented techniques today and gradually shift their data over time to the most appropriate repository.

In fact, there is often no choice about where to store objects for an application. For example, the attributes of the objects may already exist in a relational table. In addition, the application may need certain features that are not yet available in object databases such as sophisticated locking and concurrency mechanisms.

In other situations, it may be important to store some objects in a relational database and others in an object database. This would allow for "hybrid" object models, where existing data is stored in a relational database while new complex data is stored in an object database. Much of the financial industry is contemplating this kind of arrangement for building new object-oriented applications.

Benefits of Relational Mappings

There are a number of important benefits to building object-oriented applications on top of relational databases. The first is that these objects have full access to the SQL query engine. While purists argue that SQL queries violate object encapsulation, SQL remains the most powerful querying language available, and the current relational database vendors have had over ten years to expand the capabilities and optimize the performance of their SQL tools.

Next, objects in relational databases can be accessed by non-object-oriented applications. This is an important consideration, as few companies are going to go "all-objects" overnight. Keeping the data in traditional relational datastores while beginning to use object orientation for new development provides an incremental approach that will allow companies to make a smooth transition to object orientation.

CONNECTING OBJECTS AND RDBMSs: FEASIBLE BUT TIME-CONSUMING

The mechanics for mapping objects to relational databases are straightforward. In fact, many of the core ideas for object orientation come from the world of relational data modelling. The difficulty is that writing the code to perform the mapping between objects and tables is very labor intensive.

Current Data May Be More Object Oriented Than You Think

In the past, major system design efforts had two teams, one of

which pursued the functional decomposition while the other worked out the logical data model. Because structured development put an emphasis on the functional side of development, the functional team usually got its way in resolving conflicts with the data modeling team. As a result, the delivered systems met the original specifications but often lacked flexibility to accomodate changes in the business requirements.

Object-oriented development, on the other hand, takes a more data-centric view of the application. Objects are built around core sets of data, combining a set of data with a set of responsibilities for that data. In many ways, object orientation can be considered the revenge of the data modellers.

After all, it's not as if the data modellers have been sitting idle all these years. They have developed powerful techniques for modeling the data requirements of their businesses using entity-relationship diagrams. In many cases, they have defined the entities, attributes, and relationships required to run the business. In short, they have gone a long way towards building an enterprise object model.

For anyone who doubts the similarities here, try this simple experiment. Grab any book on database design and open it to a random E-R model. Now grab any book on object-oriented design and open it to a random object model. Finally, convert the notations and try to find a difference between the resulting models.

Objects Map Readily To Relational Tables

The relational algebra that forms the underpinnings for relational databases guarantees a relational database can handle any kind of data you care to throw at it. Yes, there is a performance penalty for certain kinds of highly fragmented or structured data, but the fact is that relational databases are more flexible than you might think.

At Persistence Software, for example, we created a benchmark application with 160 classes, 480 relationships, and 30,000 instances stored in Sybase. In our testing (run on a Sun IPX with 32MB RAM) the application loaded the entire data structure in little more than a minute and was able to traverse the data structure in less than a second, using object caching techniques similar to Versant. In comparing these results with the findings from similar benchmarks, such as Rick Cattell's OO1 benchmark, the loading time is 20 to 30% slower than for an OODB while the time to traverse the structure in cache is almost identical.

Mapping Tables To Objects

The translation between an E-R model of the existing tables and an object model of the corresponding objects is almost automatic. Entities become classes and relationships between entities can become relationships between classes. Views can also map to classes, allowing developers to provide "de-normalized" data for their classes.

User-Defined Types

This handles "well-behaved" data. What about user-defined types such as addresses? These can be handled by creating a table for each abstract data type and creating a foreign key mapping between the class and its abstract type members. For example, address may be a user-defined type, stored in its own table. If the employee class contained an attribute of type address, the employee table could contain a foreign key pointer to the address table.

Repeating data types can be handled in the same way. For example, an invoice might contain any number of line items. This can be represented as a one-to-many relationship between the invoice table and the line item table—a pointer in the C++ world and a foreign key in the relational world.

Class relationships can also be represented through foreign keys. For example, if the employee class is related to the department class through the relationship worksIn, this would be represented in the database as a foreign key in the employee table that pointed to the department table.

Inheritance

There are several ways to represent inheritance in relational tables. The simplest way is to create a table for each "leaf class" in the inheritance hierarchy. Each table would contain all the inherited attributes while each class would inherit attributes and methods from its superclasses. For example, if employee and customer inherited from person, there would be two tables in the database (one for employee and one for customer). employee and customer would inherit attributes and methods from person.

A second way to handle inheritance is to have a table for each class containing only the unique attributes for that class. Each subclass would also have a relationship with its superclass to get the full attribute set for an instance. In this situation, person, employee, and customer would each have a table in the database, with a one-to-one relationship between each employee instance and a particular person instance (and between each customer and a person).

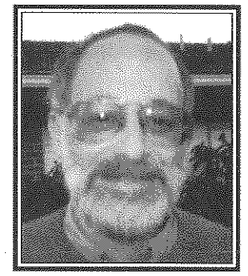
The third way to handle inheritance is to create a "supertable" in the database that contains all attributes for the inheritance hierarchy, along with a type attribute to indicate the type of class to which each row in the table corresponds. Here there would be only a person table in the database, with an attribute to indicate whether this instance was really a person, employee, or customer.

Problem: Mapping Objects To Relational Tables Takes Time

While the mapping between objects and relational databases is relatively straightforward, writing the code to implement this mapping is hard work. Today, object-oriented developers must hand-code routines to translate between each object and its corresponding table. For each class, the developer must supply methods to create, read, update, and delete the class (the C.R.U.D. operations). Each class also needs methods to set and access related objects.

Foreign keys entail not only performing cross-table lookups but also enforcing referential integrity when key values are updated and delete constraints when objects are deleted. Together, these methods can add hundreds of lines of code to each class the developer wants stored in the database. In working with our customers, we have found that developing the database interface typically consumes 20–30% of the total development time for the project, or somewhere between one and two weeks per class.

The middleware challenge



Dr. Robert Marcus

Middleware products are a new generation of tools that allow application programmers to write programs in heterogeneous distributed environments without having to deal with the intricacies of underlying system software. During the last few years, there has been a slow but steady development of this type of tool. Middleware has not gotten the same attention as the new GUIs, databases, languages, and operating systems but it could prove more important in the development of corporate computing environments. The integration between the new middleware products and object-oriented technology will be very important for future distributed systems.

The purpose of this article is to draw attention to what may be an upcoming revolution in future delivery systems and the major implications for possible business process reengineering. Some layers and sublayers will be defined to clarify the distinctions in middleware functionality. Then there will be a brief discussion of the client–server vs. asynchronous messaging paradigms and the implications for the Object Management Group's (OMG) Object Request Broker (ORB). An overview of some of the asynchronous messaging and RPC products follows. An end user's group called the Corporate Facilitators of Object-Oriented Technology (CFOOT) has been started to keep corporate technologists abreast of trends like middleware and some information about this group is provided. Finally, some future challenges for developers and business process planners are listed.

TWO REFERENCE MODELS

I have found two reference models to be helpful when trying to categorize the middleware products. A first step is to locate middleware in the spectrum of tools that facilitate software development.

One workable classification is:

- **outerware**—helps end users develop programs (e.g., macro and visual programming languages)
- **middleware**—helps application programmers develop programs (e.g., messaging products described below)
- **innerware**—helps system programmers develop programs (e.g., object layers on top of the operating system).

There are several different types of middleware including data access, mail messaging, user interface, and interapplication communication. This article will focus on middleware for interapplication communication. Middleware can be further categorized by the level of abstraction provided to the application programmer. For interapplication communication in distributed systems, one possible partition with corresponding application programming interfaces (APIs) is:

- **upper middleware**—mediator API encapsulates the receivers of messages (e.g., software brokers, traders, mediators, and intelligent agents)
- **middle middleware**—object API encapsulates the receiver's implementation (e.g., object request brokers)
- **lower middleware**—generic API encapsulates underlying operating systems and network. (e.g., remote procedure calls and messaging products)

The basic principle in distributed systems middleware is to hide the complexity of network communication protocols from the application programmer. For example, remote procedure calls (RPCs) are used as the basis of systems like the Open Software Foundation's (OSF) Distributed Computing Environment. An even higher level of abstraction can be achieved by using object-level APIs such as the Object Request Broker from the OMG. An ORB can hide the differences between the underlying RPC or messaging implementation. The mediator level may be necessary to provide interoperability between different ORB implementations or construct large-scale heterogeneous environments. The International Standards Organization's (ISO) Open Distributed Processing (ODP) group's Trader is an attempt to specify standards for this type of functionality.

PEER–TO–PEER VS. STRICTLY CLIENT–SERVER

One of the most important issues in distributed interapplication communication middleware is the choice of the peer–to–peer vs. strictly client–server paradigm. This is sometimes posed as asynchronous messaging vs. remote procedure calls. However, it is possible to implement peer–to–peer interactions on top of RPCs by using multiple threads (e.g., Sun's ToolTalk on top of their transport-independent RPC). It is also easy to build synchronous client–server applications on top of asynchronous messaging.

From a pragmatic viewpoint, when components of a distributed system are relatively independent or run in heterogeneous distributed environments, an asynchronous peer–to–peer approach seems more suitable as the foundation paradigm. This allows capabilities such as guaranteed delivery, priority message queuing, notification, and broadcasting to be supported. For distributed application components working in a local environment,

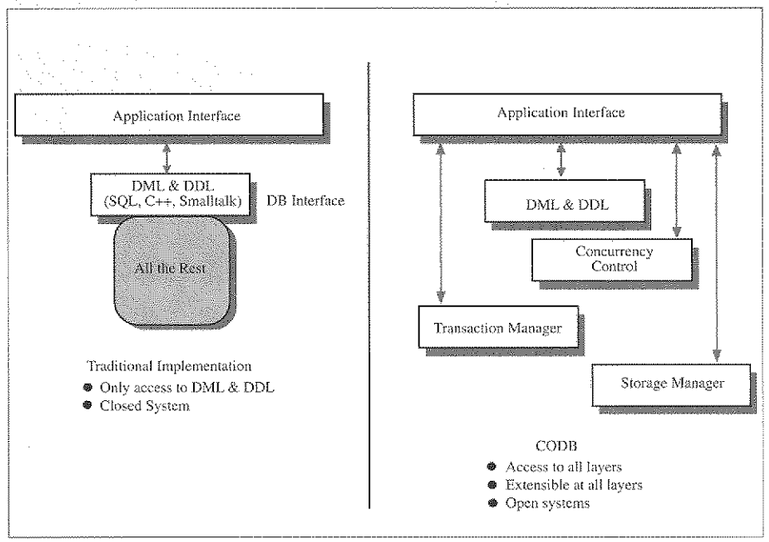


Figure 2. Traditional vs. CODB implementations.

they are generally secondary to language interface support. DOBs approach the problem from the traditional database perspective and concentrate on adding objects to the typical database mechanisms. These systems can be viewed as an evolution of the current RDB, with classes and objects replacing tables and rows.

In contrast to PLEs, DOBs place more emphasis on multi-user transaction capabilities than language integration. DOBs offer more flexible locking protocols and are designed to be used in multi-user environments. For example, PLEs often offer no locking at all or page-level locking. DOBs provide locking at the object level. This feature makes the DOB a more complex system but offers the flexibility needed to support concurrent use in commercial applications such as financial trading.

Component Object Databases

Finally, there are component object databases, which use an entirely new architecture based on object technology. This architecture adds unique extensibility and flexibility to the capabilities of PLEs and DOBs. For example, the object-oriented architecture of CODBs enables the use of multiple low-level implementations of object storage in a manner transparent to the developer.

Another unique advantage is that the extensibility of CODBs enables new technology to be integrated without changing systems. For example, when new technologies such as special hardware and software to support audio and video become available they can be seamlessly integrated into the CODB and immediately used by applications without having to recode or recompile any application logic.

CODBs build on top of both PLE functionality and DOB functionality by using objects to change the implementation architecture of the database. PLEs and DOBs are implemented as large, closed systems similar to the traditional RDBs shown in Figure 2.

CODBs, in contrast, are implemented in modular component class libraries. These component class libraries can be accessed and extended independently, which makes CODBs flexible, accessible, and extensible at all layers of the system (as shown in Fig. 2). For example, the storage management component of an object can be changed independently of the application's use of the object. This flexibility enables the developer to make changes to the storage model of the object without affecting the logic of the application. An excellent separation is created between the *use* of objects in an application and the *source* of objects on disk. Performance bottlenecks in database applications can thus be addressed without recoding the application or changing the application design.

Another important benefit of the CODB approach is that new requirements can be added to an existing system by either the vendor or the developer. In this sense, the CODB is truly an enabling technology because customers can extend the system without vendor assistance, thereby responding to changing requirements more rapidly than is possible when vendor release cycles are involved. One obvious example of this is multimedia, where rapid advances are being made in technology for storing and retrieving digital video, voice, and other media forms. With a CODB, a new class of storage manager to handle digital video streams can be added to an existing system by the vendor or customer. Applications can use this storage manager without adding code, recompiling, or relinking, enabling existing applications to take advantage of the new technology—a true competitive advantage for organizations.

CONCLUSION

Object database technology is being used more and more to create high value applications for organizations. Object databases can be roughly placed into three categories. Persistent language environments focus on tight integration with a compiler to ease programming for the low level coder. Databases of objects provide a traditional database architecture using classes and objects as the descriptive mechanism replacing tables and rows. Component object databases use object technology to modularize the system and enable components to be freely mixed and extended to accommodate a wide variety of applications. Products in all of these categories are providing significant value to developers and users. ≡≡

Tim Andrews is Chief Technical Officer at ONTOS Inc. and one of its primary designers.He has a background in object technology, database implementation, and technical marketing. He can be reached at ONTOS, Three Burlington Woods, Burlington, MA 01803, 617.272.7100; fax: 617.272.8101.

The code that results from hand-coding object interfaces is generally not reusable across projects and certainly not portable between databases. In addition, each developer tends to use different standards for locking data and maintaining referential integrity and delete constraints.

TOOLS AUTOMATE INTEGRATION BETWEEN OBJECTS AND RDBMSs

While relational databases are in many respects well suited as repositories for most object data, there are integration and performance issues involved. To a large extent, these problems can be solved by application development tools that automate the generation of the database interface and speed runtime performance through object caching.

Database Interface Builder Automates Coding of Data Access Methods

Developers building graphic user interfaces today are accustomed to having tools that help automate their task. For example, there are a number of libraries available that provide object-oriented interfaces to the X Window System. There are also a number of graphic interface builders that automatically generate interface methods based on a screen mockup.

> *Another benefit of object databases is their tight language integration.*

Similar tools are available for developers building database interfaces. As in the GUI world, there are libraries that provide an object-oriented interface to the underlying database. This kind of library allows the developer to send SQL queries to the database and get back tuple objects as the result of the query.

This approach simplifies interaction with the database but still leaves the developer with the task of hand-coding the translation routines between the generic tuple objects and their application objects. One example of an object-oriented class library for relational databases is db++ (available through the Qualix Group).

There are also tools for generating database interfaces automatically, much like the graphic interface generators available today. These tools take an application object model as input and generate "data smart classes" as output. These classes contain all the methods needed to read and write themselves to the database. In addition, they have methods to set and access instances of related classes.

Our product, Persistence, is an example of a database interface builder. Once the developer has specified the object model for an application, the Persistence Database Interface Generator produces a custom interface for each object, complete with create, read, update, and delete methods and methods to set and access related objects (e.g., how to perform foreign key lookups). Because each object "knows" how to store and retrieve itself, the developer can treat the relational database as if it were object oriented.

Caching Can Speed Performance For Complex Data Types

A second issue in working with relational databases is their performance. This can be split into two separate issues: speed in retrieving the data and speed in traversing highly structured data (e.g., bill of materials or multimedia documents).

To improve speed in retrieving object data, it may be necessary to denormalize the database tables to achieve adequate performance. Another option is to create a view that represents a join between several tables and then map that view to an object.

To provide speed in traversing object structures, object databases such as ObjectStore from Object Design cache objects in memory. With an object cache, a large, complex object can be brought into memory and then traversed there rather than having to go back to the database each time the data is accessed.

But, object data from relational databases can also be cached with similar performance benefits. For example, Persistence automatically performs database locking and object caching for objects accessed within a transaction. In our benchmarking, we have found that object caching can speed relational database performance by a factor of ten for data-intensive applications.

The Role For Object Databases

This is not to minimize the role and potential market for object databases. There are fundamental differences between relational and object-oriented databases. As Esther Dyson says, "relational databases are good for data, object databases are good for structure." As companies become more comfortable with the object paradigm, they will find more and more valuable applications for object databases. Yet for companies just starting with objects, or for companies with significant investments in relational databases, it is more practical to start with applications built on top of existing database management systems.

The most important reason for using object databases is to achieve adequate performance for handling complex object structures. Object databases store information about the structure of their data directly while relational databases don't. For example, databases like ObjectStore store pointer information rather than foreign keys. For highly structured data, this can improve performance in retrieving and storing the data.

Another benefit of object databases is their tight language integration. By sparing the developer the difficulty of flipping between their procedural language and the database's declarative language, object databases offer faster development. On the other hand, tools are becoming available to provide a tight language interface between the developer's object model and relational databases as well.

SUMMARY

Object popularizers have so far focused too much on describing the wonders of this new technology and too little on offering practical advice about how to get started. Like the pioneers of old, they sometimes seem more interested in holding audiences

spellbound with tall tales than in building roads so that others can go where they have gone.

While object orientation offers significant benefits to software developers, these benefits are hollow if they first require companies to abandon their existing information infrastructure to achieve them. Thus, the first order of business for getting corporate acceptance of objects is to show how to build object applications on top of existing data. Now that enabling products are appearing on the market, many previously hesitant companies are moving more aggressively to adopt object technology, as shown by the growing attendance at shows like Object Expo, ObjectWorld, and OOPSLA.

Over time, adoption of object technology will drive a steady migration to more powerful databases. This will come as developers use the power of object encapsulation to create more sophisticated object designs. As this happens, new applications will use a hybrid approach to access existing data in relational databases while storing more complex object data in object databases. ≡ ≡

Christopher Keene is president of Persistence Software, Inc. of San Mateo, CA. Before founding Persistance Software, Chris was a manager with McKinsey & Co., worked in marketing at Ashton-Tate, and was a software engineer at Hewlett-Packard. He can be reached at 415.341.7733 or ckeene@persistence.com.

2. Miller, M. Xanadu Operating Company, private conversation on software pay-per-use, July 1992.
3. Mori, R. and M. Kawahara. Superdistribution: An Overview and the Current Status, TECHNICAL RESEARCH REPORTS OF THE INSTITUTE OF ELECTRONICS, INFORMATION, AND COMMUNICATION ENGINEERS, 89[44], 19xx.
4. Hemnes, T.M.S., Ropes, and Gray. SOFTWARE REVENUE GENERATION IN NETWORK ENVIRONMENTS, Massachusetts Computer Software Council Annual Legal Update Program, November 1992.
5. Simmel, S.S. KALA—INTERFACE REFERENCE PART I: KALA FACILITIES, Rev. 2.0, Software Version 2.1, Penobscot Development Corporation, Arlington, MA, 1991.
6. Simmel, S.S. KALA—MAIN CONCEPTS, Revision 1.0, Penobscot Development Corporation, Arlington, MA, 1990.
7. Simmel, S.S. and I. Godard. The Kala basket—a semantic primitive unifying object transactions, access control, versions, and conftgurations, PROCEEDINGS OF OOPSLA '91, October 1991.
8. Simmel, S.S. and I. Godard. Objects of substance, BYTE MAGAZINE 15[1], 1992.

Sergiu S. Simmel is cofounder of Penobscot Development Corporation of Arlington, MA. He has been involved in the Kala project for the past six years as a codesigner, author, communicator, implementor, and business manager. His background covers CASE systems, object-oriented technologies and languages, software engineering, and databases. He can be reached at 617.646.7935; fax: 617.646.5753.

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 Dylan Smith, 588 Broadway, Suite 604, New York, NY 10012; fax: 212.274.0646.

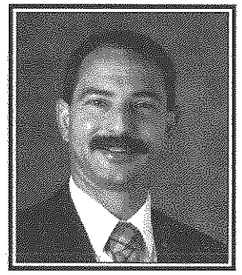
February 1–4 and 4–5, 1993 OOP '93/C++ World Munich, Germany Contact: 212.274.9135	February 15–19, 1993 MADACON '93 San Diego Hilton Beach and Tennis Resort Contact: 408.253.2765	February 21–26, 1993 Software Development Spring '93 Santa Clara Convention Center, Santa Clara, CA Contact: 415.905.2319	March 8–11, 1993 X-World Marriot Marquis New York, NY Contact: 212.274.9135
March 8–11, 1993 TOOLS EUROPE 93 Versailles, France Contact: +33.1.45.32.58.80	March 30–April 1, 1993 Object Technology '93 Cambridge, England Transfer +44.491.410222	April 19–23, 1993 Object Expo Hilton Towers New York, NY Contact: 212.274.9135	April 22 & 23, 1993 International Symposium & Exhibition on Object Technology: Methodologies and Tools Frankfurt, Germany Contact: +49.69.52.19.82.

Note to our readers:

To make it easier to save and protect your copies for back reference, the HOTLINE *has been redesigned to fit into a standard three-hole punch looseleaf binder.*

Customized HOTLINE *binders hold two volume years and can be purchased for* $15 *(including shipping and handling) by calling* 212.274.0640.

A description of object databases



Tim Andrews

One of the questions most frequently asked of people in my position is "What is an object database and what is it good for?" The next few columns will be devoted to answering these questions. I'll start with a description of the technology.

The object database represents a new database technology rapidly gaining acceptance in the marketplace. There is some confusion surrounding the impact of object database technology and the segmentation of the products available. The object database represents the convergence of two substantial technology streams: object-oriented programming languages and database management systems. A complete understanding of object database technology requires in-depth knowledge of both object-oriented programming languages (OOPLs) and database management systems (DBMSs). In fact, one of the challenges of trying to explain object databases is that most technical people have a heavy bias in either the DBMS or OOPL direction and, as a result, are missing an important body of knowledge needed to properly interpret the technology.

"Objects" were originally a language phenomenon, first appearing in the language Simula in 1967. Today objects are embodied in C++, Smalltalk, and a host of other languages. Database management systems originated in the 1960s with the IMS system from IBM and have evolved into relational database products such as Oracle, Ingres, and Sybase and PC database products such as Paradox, Dbase, and FoxBase.

Products in the object database market fall roughly into three categories: persistent language environments (PLEs), databases of objects (DOBs), and component object databases (CODBs). These categories are more or less functionally layered. In other words, databases of objects provide a superset of the functionality of persistent language environments, and component object databases add additional capabilities to databases of objects (Fig. 1).

PERSISTENT LANGUAGE ENVIRONMENTS

A PLE's primary function is to facilitate the storage and retrieval of objects from within a given language environment. This category is focused on developers who use a specific language with the products generally very closely bound to the language, usually C++ or Smalltalk. These products are used more in single-user environments where a single developer creates an application, e.g., by converting an existing C++ application. CAD applications, in particular, are well suited to this approach in the workstation marketplace.

One reason for the development of PLEs is the desire of programmers using an OOPL to store and retrieve objects from a secondary storage device such as a hard disk. Since the objects in an OOPL are created and managed by the language environment, support must be added if the programmer wants to save objects for use in a subsequent session. The programmer has a range of choices in this case: write code to save and recall objects from disk, use an existing DBMS, such as a relational database management system (RDBMS), or use an ODB. A PLE can increase the productivity of the developer in this case by eliminating the need to write special code to store and retrieve objects.

Databases of Objects

Databases of objects (DOBs) are systems that provide, in addition to the basic persistence of PLEs, storage and retrieval of objects from multiple languages along with more traditional database features such as concurrent access, 4GLs, and higher-level tools for developing database applications. Currently, the languages supported are C++ and Smalltalk, with LISP and other languages garnering some support. These systems allow for the development of multi-user applications. In addition, DOBs that work with graphical programming tools are accessible to a much broader group of programmers.

In the discussion of PLEs, the features normally associated with database systems, such as multi-user access and reliable transactions, were not mentioned. PLEs approach the problem from the language perspective; if database features are present,

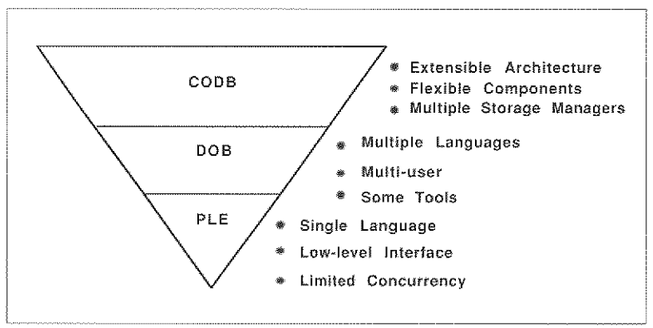


Figure 1. Product catagories.

hotline on
OBJECT-ORIENTED
technology

Back issues

All back issues of the HOTLINE are available. Please call 212.274.0640 for details.

Vol. 4, No. 3/January '93 ≡ Gauntlet implementations: the 5GL object-oriented challenge ≡ Infrastructure for a new economics of software ≡ You can have your objects and be relational too! ≡ A description of object databases ≡ The middleware challenge

Vol. 4, No.2/December '92 ≡ Zero-cost portability ≡ Design by contract: building bug-free O-O software ≡ Object interest group: phase two ≡ Towards a framework for software ROI ≡ Amziod "objects"

Vol. 4, No. 1/November '92 ≡ Combining object technology with data standards for the next industrial revolution ≡ Constant quality management ≡ Evolving markets for software components ≡ The quest for value ≡ Reviewing OOSE: a use case-driven approach

Vol. 3, No. 12/October '92 ≡ ROI: development environments for the lifecycle ≡ Selecting the right object-oriented method ≡ Choosing an object-oriented language ≡ Object database technology: who's using it and why? ≡ Objects and reuse

Vol. 3, No. 11/September '92 ≡ Developing strategic business systems using object technology ≡ Object training: harder than it looks ≡ Object-oriented ROI: extending the CRC across the lifecycle ≡ What TQM means for OT

Vol.3, No.10/August '92 ≡ Object technology: toward software manufacturing ≡ Return on investment: software assets and the CRC technique ≡ Object-oriented technology in Japan ≡ Providing commonality while supporting diversity

Vol.3, No.9/July '92 ≡ OOD: Research or ready ≡ Enterprise modeling: an object approach ≡ OMG's 18–24 month view ≡ Design for object-oriented applications: a CASE for wishful thinking. . .

Vol.3, No.8/June '92 ≡ Business in the Information Age ≡ From data modeling to object modeling ≡ How frameworks enable application portability ≡ Interview with Vaughan Merlyn

Vol.3, No.6/April '92 ≡ Thinking the unthinkable: reducing the risk of failure ≡ Mitigating madness with method: first establish what you value ≡ Championing object technology for career success in the 1990s ≡ Objects and actions in end-user documentation

Vol.3, No.5/March '92 ≡ TA large-scale users' assessment of object orientation ≡ Report on the Object-Oriented COBOL Task Group ≡ Interview with K.C. Branscomb

Vol.3, No.4/February '92 ≡ The big prize: acceptance of O-O by the MIS community ≡ Retrospective: 1991—the year it all changed ≡ Making the transition to O-O technology ≡ Interview with Beatriz Infante

Vol.3, No. 3/January '92 ≡ Enterprise object modeling: knowing what we know ≡ Adopting objects: pitfalls ≡ Adoption rate of object technology: a survey of NSW industry

Vol.3, No. 2/December '91 ≡ Accepting object Technology ≡ Adopting objects: a path ≡ Incorporating graphical content into multimedia presentations

Vol.3, No. 1/November '91 ≡ Leading the U.S. semiconductor manufacturing industry toward an object-oriented technology standard ≡ Coping with complexity: OOPS and the economists' critique of central planning ≡ Choosing Object Technology: What's the object? ≡ OOP: the MISsing link

Vol.2, No. 12/October '91 ≡ A modest survey of OOD approaches ≡ What is a "certified" object programmer? ≡ Perspective: investing in objects today ≡ Object oriented in Melbourne, Australia ≡ The Object Management Group

Vol.2, No. 11/September '91 ≡ From applications to frameworks ≡ Report on the Object-Oriented COBOL Task Group ≡ Getting started with object technology: efffectively planning for change ≡ Object statistics on the way ≡ On objects and bullets

Vol.2, No. 10/August '91 ≡ Distributed object management: improving worker productivity ≡ Getting the best from objects: the experience of HP ≡ APPLICATIONS: EC employs object technology ≡ CAPACITY PLANNING: Fiddling while ROMs burn

Vol.2, No. 9/July '91 ≡ Multimedia is everywhere! ≡ Developing an object technology prototype ≡ Object-oriented capacity planning ≡ How OOP has changed our developmental lifecycle ≡ Modularization of the computer system

Vol.2, No. 8/June '91 ≡ Domain of objects: the Object Request Broker ≡ Object-based approach to user documentation ≡ Report on the Object-Oriented COBOL Task Group ≡ Do we need object-oriented design metrics?

SUBSCRIBE NOW TO THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY— DON'T MISS ANOTHER VALUE-PACKED ISSUE!

❑ *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)*

(outside US add $30 per year for air service)

Back issues @ $25 each *($27.50 foreign):*

Vol.2, Nos. _____ Vol.3, Nos. _____

Vol.4, Nos. _____

❑ Phone/fax order
Call Subscriber Services at 212.274.0640
or fax this form to 212.274.0646

❑ Bill me

❑ Check enclosed
Make check payable to the HOTLINE and mail 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)

❑ Credit card orders
❑ MasterCard ❑ Visa ❑ AmEx

Card# Expiration Date

Signature

Name

Title

Company/Mail Stop

Street/Building#

City/Province

ST/Zip/Country

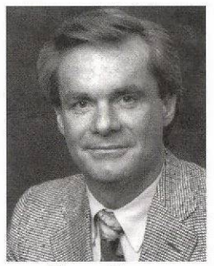
Telephone

D2KC

hotline on
OBJECT-ORIENTED
technology

VOL. 4, NO. 4 THE MANAGER'S SOURCE FOR TRENDS, ISSUES & STRATEGIES FEB. 1993

The Need For Quality



Greg Pope

While the United States is the unequivocal leader in the global software market, it is no secret that our software is infested with bugs. The existence of this anomaly is possible only in the absence of serious international competition. Like the automobile industry in the 1950s and 1960s and the semiconductor industry in the 1970s and 1980s, the American software industry has used its pioneering innovations to sustain a prolonged period of unchallenged prolific growth.

This period is now over. Foreign software developers are continuing to produce more innovative products with high-quality code, forcing the U.S. software industry to reevaluate its entire software development methodology. SAP, for example, the German-based MRP-II software vendor, is Europe's largest manufacturing software vendor and one of the emerging COMMS players in North America and worldwide. If quality does not become a priority, foreign software vendors like SAP will devour the US's worldwide market share.

The issue of software quality transcends the domain of American economic competitiveness and impacts everyone living in an industrialized society. As heirs of the technology revolution, we unwittingly interface with several million lines of software code everyday. Making a phone call, for instance, depends on the existence of over 500,000 lines of reliable software code. Digital alarm clocks, stereo systems, traffic signals, aviation equipment, medical devices, televisions, and car braking systems all rely on quality software programs. Software is also a primary tool for writing letters, calculating financial reports, and designing electronic circuits and buildings.

The need for reliable, bug-free software is no longer an issue that belongs exclusively in the software testing laboratory. It is an issue affecting our economic position in the global marketplace as well as our everyday safety and well-being.

TO ERR IS HUMAN

The development of software applications is a complex and arduous task, sometimes requiring several hundred thousand lines of software code for a single application. Because software development is still largely a human endeavor, it will, by its nature, generate errors. If these errors are not detected and corrected, they can cause the software to malfunction. Depending on the specific application, a software malfunction can be a minor nuisance or, in the case of systems controlling hazardous materials, result in injury or death.

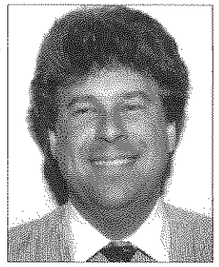
THE SOFTWARE DEVELOPMENT REVOLUTION

The demand for efficient and thorough software testing tools is magnified by the advent of computer-aided software engineering tools (CASE), software reengineering, object-oriented program-

continued on page 4

IN THIS ISSUE

FROM THE EDITOR



Early last year, surveys indicated that 40% of all United States IT organizations would start pilot or full-bore development efforts using objects. Telling, however, was that 60% of mainframe IT shops expected to start using or exploring objects last year. Informal project surveys support the direction, if not the magnitude, of industry growth. Often these same shops were downsizing, experimenting with client-server architecture, and looking for ways to deliver applications to serve very dynamic, short-fused, short-lived market opportunities. Against this backdrop, Mr. Tom Love, noted industry consultant and president of OrgWare, brings us his annual commentary on events in the object industry. While the main event in 1991 was seeing the term *object-oriented* in the business press, Mr. Love points out that 1992 was the first commercial year for the technology. Object technology addresses critical needs for the large IT shop. The significant uptake of this technology by such organizations suggests that 1993 will be an even stronger year for the industry than was 1992.

Continued growth depends on setting reasonable customer expectations and meeting those expectations while the prospective customer is focused on this industry. Setting reasonable expectations is a function of accurate communication between vendors and customers. How vendors communicate to customers about themselves and their products is vital. For example, recent press materials distributed by ODBMS vendor Objectivity gave incorrect impressions of the market share of several of its successful competitors—companies like ODI, ONTOS and Servio that are, based on available evidence, actually outperforming Objectivity in the marketplace. Distributing accurate product information is a first step to setting reasonable market expectations and spurring market growth. Let's keep ourselves on track.

Meeting expectations and growing the market also depend heavily on delivering quality products. Software quality is that elusive concept that some define as the lack of defects. We call them bugs, but remember that the term *bug* originated with finding a moth stuck in the register relays of the Iliac! Software defects are more sinister and far less pretty than a moth. Others define software quality as possessing characteristics that fascinate the customer. Being defect-free is a necessary prerequisite but is not itself sufficient. By this definition, a quality product is not only true to specifications but anticipates its users' interests, work challenges, and creative insights. It is pleasing to use, supportive of the worker, and stimulates creative business thinking—all while helping us get our jobs done. Meeting market expectations depends on both aspects of quality. For growth of this market to accelerate in 1993, object technology products need to do what they're supposed to and what the customer needs. This is equally true for commercial products and business applications.

Both aspects of quality are a function of our approach to software development. The process of producing defect-free software starts with developing a sound understanding of user requirements, depends on rigorous testing, and remains critical throughout the entire field life of the software product. Developing software that fascinates also depends on our understanding of the user's needs and expectations, but effecting fascination is more a function of the professional developer's ability to innovate within the user's problem space than a result of creativity with the tools of the software trade. From various viewpoints, our authors this month examine the prerequisites to delivering quality software.

Mr. Greg Pope of Tiburon Systems writes about rigorous product testing as a key part of the quest for quality software. Mr. Pope observes that the traditional global leadership position of the U.S. software industry is in serious jeopardy. Anyone who saw the influx of superb French-built O-O software tools at industry conferences during 1992 would agree. Japan has taken a sudden and fierce interest in object technology, which, added to their strength in applied fuzzy logic, could manifest itself in a powerful new generation of consumer products. I have previously mentioned India and Ireland, where low-cost labor and high-talent people combine to deliver a competitive software manufacturing environment. As Mr. Pope observes, we may have been the pioneers, but we cannot survive on naive anticipation of unchallenged growth.

Mr. Pope's particularly thought-provoking observations on the impact of software on our lives caused me to inventory the software-dependent services on which you and I are completely dependent. Consider these additions to his list: banking back office and ATM networks; stock market automated trading; E911 emergency dispatch for medical, fire and police; electrical power grid control; cash registers and credit card transaction processing. How would a day in complete darkness, isolated from your money, customers, business associates, and emergency services sound? Relaxing? Perhaps we do care about software quality!

Why, then, do commercial and corporate development managers and developers place testing so low on the real priority list?

Mr. Pope's comment on market complacency is painfully accurate. How many of us customers put up with exactly the scenario he describes: encouraging vendors by purchasing defective and incomplete products; tolerating vendors who expect us to spend days strip-testing their problems on our machines, at our expense, on our time—as though the vendor's problem is our most important task? Is your firm dependent on Microsoft Windows? In my office we are. When Windows 3.1 shipped, our postscript printer failed. Word for Windows would not print envelopes correctly. The demonstration projects in Microsoft Project 3.0 would hang the printer. After well over 20 hours on the phone—mostly at our expense—performing various regression tests with Microsoft technical support staff, one deter-

much of the complexity in a large system is hidden from programmers, leaving them more time to concentrate on making programs work as designed. . . .

Tech insider: Object-oriented software—of paradigms and pizzas, Curtis Franklin, Jr., **VAR BUSINESS**, 10/92

. . . Not only is object modeling a potent technique for more effective programming, but it is a very powerful communication tool that can enhance understanding and cooperation in complex team efforts. Not that all of the examples above have described objects, event and operations in terms of the overall team's business focus, not in terms of the concerns of any specific team member. The graphic artist can work out issues of color and shading, the programmer can fine tune his windows and widgets, but the multimedia team as a whole needs to fully understand and agree upon how the overall drama is going to unfold. Properly employed, an object modeling method can greatly facilitate this process, and cut your costs and time to market.

Object modeling can ease multimedia development, Robert E. Damashek, **COMPUTER PICTURES**, 10/92

. . . "When you have to support a lot of vendors' products, object-orientated software pays off big time," [NeoCAD's Bruce Talley] points out. "You can keep the same underlying data structures and add personality modules." . . . .

Start-up crafts device-independent FPGA tools, John Haystead, **EDN NEWS**, 10/22/92

. . . "In the object world you start by defining classes," explained Lanny Lampl, a technical consultant in Levi Strauss' Information Resources Group. "You have to parcel out the responsibilities of each object and decide how classes will interact with each other." Carrying out an object-oriented analysis turned out to be harder than switching to SmallTalk. "The syntax of the language is not the big thing," Lampl said. "The important thing is learning how to think about objects."

Levi Strauss cuts client/server pattern, Jean S. Bozman, **COMPUTERWORLD**, 11/16/92

MODELING

. . . The "dimensions" of encapsulation and reactiveness provide a framework for distinguishing modeling paradigms based on objects from deductive reasoning paradigms based on logic. Object-oriented systems are both encapsulated and reactive while logic programming systems are nonencapsulated and nonreactive. Concurrent logic programs are reactive but not encapsulated. Advocates of concurrent logic programming feel that this compromise combines the advantages of logic with the power of reactiveness, while skeptics feel that the compromise falls between two stools by compromising the integrity of logic without providing a systematic framework for programming in the large. . . .

Dimensions of object-oriented modeling, Peter Wegner, **COMPUTER**, 10/92

APPLICATIONS

. . . [Bob Zurich, vice president of research and education for Infinity Systems Corp] says the three biggest mistakes IS shops make regarding object-oriented application development are "building stuff that's not really reusable, overdoing it, not getting into it not." Like most OOP experts, Zurich recommends an incremental approach to OOP. One thing many IS shops still don't realize is that, in OOP environments, Prototypes don't have to be discarded when the real application is coded. Instead, prototypes go through many iterations and end up becoming the production code. [Wayne Adams of Southern California Gas] uses Enfin in this type of environment. He says prototyping often works so well that even the project done to evaluate Enfin wound up being used in production. . . .

Windows drives OOP on the desktop, Lee Thé, **DATAMATION**, 11/1/92

THE BUSINESS OF OBJECTS

The market for object-oriented software systems in the US and Europe will grow from $865 million now to well over $4 billion in 1997. But a new report by the London-based Ovum Ltd. market research firm says this growth is slower than expected because "the worldwide recession has affected company investment plans and object-oriented products are not taking off as quickly as previously assumed." . . . .

Object-oriented market to grow steadily: **OVUM, SOFTWARE INDUSTRY REPORT**, 11/2/92

Unlike a number of object-oriented programming languages, C++ does not reify the notion of a class. Stated in simple terms, classes in C++ are not represented by objects. Objects are instances of classes, but there is no way to write a class that has other classes as its instances. This is not usually a problem, and it certainly keeps with the general design philosophy of the language. C++ is designed to do type checking at compile time whenever possible. Generally, there is no need to have objects representing types that supply run-time information about the type. But occasionally it would be nice to obtain, store, and muck about with information about a class of objects, especially when attempting to instrument a program. . . .

C++ Advisor: Global static, Jim Waldo, **UNIX REVIEW**, 11/92

DESIGN

. . . Object-oriented analysis and design methodologies are rapidly evolving, but the field is by no means fully mature. None of the methodologies reviewed here (with the possible exception of Booch OOD) has—as of this writing—achieved the status of a widely recognized standard on the order of the conventional methodologies of Yourdon and Constantine or DeMarco. Object-oriented methodologies will continue to evolve, as did conventional methodologies before them, as subtler issues emerge from their use in a wide array of problem domains and project environments. As discussed above, three areas—system partitioning, end-to-end process modeling, and harvesting reuse—appear to be especially strong candidates for further development work. In the meantime, adopters of current object-oriented methodologies may need to develop their own extensions to contend with these issues or, alternatively, limit application of the methodologies to problem domains where these issues are of lesser importance. . . .

Object-oriented and conventional analysis and design methodologies, Robert G. Fichman and Chris F. Kemerer, **COMPUTER**, 10/92

OOP is inclusive, just as structured programming was two decades ago. It differs, however, from structured programming's traditional association with functional design methods such as functional decomposition, dataflow diagrams or data structure design. In OOP, objects are first categorized into classes and organized hierarchically according to their dependency and similarity. Each class comprises a set of attributes reflecting the objects' generally static properties and a set of routines (in Smalltalk, methods) that manipulate these attributes. Then relations between classes, such as inheritance, are designed. . . .

Object-oriented computing, David C. Rine and Bharat Bhargava, **COMPUTER**, 10/92

. . . The combination of inheritance, redeclaration, polymorphism, and dynamic binding shields much of the power and flexibility that result from the use of the object-oriented approach. Yet these techniques may also raise concerns of possible misuse: What is to prevent a redeclaration from producing an effect that is incompatible with the semantics of the original version—fooling clients in a particularly bad way, especially in the context of dynamic binding? Nothing, of course, no design technique is immune to misuse. But at least it is possible to help serious designers use the technique properly. . . .

Applying "Design by Contract," Bertrand Meyer, **COMPUTER**, 10/92

. . . Although it's nice that operating systems are becoming object-oriented for the user, there's no doubt that maintaining backward compatibility with a straight C API brings with it an inherent complexity. Object management needs to be integrated much more smoothly into the operating system services and made to fit naturally with object-oriented languages. In effect, you want the operating system support for objects to be as transparent as support for memory allocation and deallocation, file services, and so on. The approach must be sufficiently general that it can accommodate a range of languages, not just C++ and Pascal. There will always be a place for interpreted languages such as Smalltalk and Actor, and I hope that future object-oriented operating systems will make cross-language sharing of objects a reality.

Polymorphism unbound, Zack Urlocker, **WINDOWS TECH JOURNAL**, 10/92

. . . Object-oriented software takes the spotlight off actions and puts it onto data. Think of it this way: In procedural languages, data are passed around the program. Bouncing from step to step in the process. In an object-oriented system data stay put while messages telling them what to do are sent hither and yon. It sound like a rather academic distinction until you realize that this new way of doing things forces programmers to rethink their approach to programming. More important, focusing on data means

mined support technician actually replicated the problem on a stripped-down system at Microsoft. When the alleged fix shipped a few months later, we discovered to our horror that the problem had actually been made worse: now the workaround we originally had been given no longer worked, and neither did the new driver. Back to ground zero! A spicy letter to Mr. Bill Gates brought phone calls from a new support person. By this time, Microsoft had lost all records of our earlier phone calls and had no record whatsoever of the postscript driver problem. After weeks of fruitless efforts to reach this new support person, we were told by letter that we should—once again—start with a strip-test of our machine. This time, Microsoft did not have the resources to test "our" problem. It's your turn, IBM.

We ourselves have unwittingly contributed to this problem by continuing to purchase product upgrades. We need the software badly enough that we accept untested and defective products. The idea of Windows is excellent but the implementation is harming each and every one of us whose work is inhibited by software defects. With tools like those described by Mr. Pope, there is positively no excuse for continued distribution of ill- or untested software products.

Shifting focus upstream, Mr. Richard Dué discusses Object Engineering, his concept for a software development process that uses object orientation from the enterprise level through to construction. Mr. Dué observes that object technology has followed a bottom-up course, from development tools to design to analysis. We do know that development of high-quality business applications depends on a solid start at the enterprise level, not just on slick development tools and application-oriented analysis. Working out the interaction of such concepts as contracts, layered business object models from enterprise through implementation, and scenario modeling will be the challenge of 1993 for Mr. Dué, Mr. Hendersen-Sellers, and others looking to raise object orientation to the enterprise level. The challenge is intensified by the fact that Dué really is taking development concepts and trying to raise them to the enterprise level—design by contract, class libraries, and use cases. How this delivers a shared base of business objects and accurately captures business semantics, while avoiding the pitfalls of traditional information engineering, is yet to be determined. The tie between the business of business and business objects themselves will facilitate or break application fitness, flexibility, utility, and return on investment. Go forth!

Mr. Timothy Andrews of ONTOS focuses his column this month on the movement toward a manufacturing model for software. Mr. Andrews picks up the theme of Mr. Brad Cox and others: the component approach to software will achieve benefits similar to what we have experienced in the computer chip and other interchangeable parts manufacturing processes. How will we achieve the manufacturing model? Mr. Pope has identified part of the problem: changing customer attitudes and eliminating complacency. Mr. Dué has struck a second chord with Mr. Bertrand Meyer's concept: design by contract for rigorous interface specification and business components that link the enterprise-level need to the implementation-level part. Mix in standard application program interfaces, as promoted by the Object Management Group's Common Object Request Broker Architecture (OMG CORBA), and a manufacturing model for distributed object computing begins to emerge from the soup.

Ms. Patti Dock of OrgWare directs her column to the cultural changes required for a successful transition to object technology. Returning to one of Mr. Pope's themes, we are talking about changing internal attitudes - not just about testing and product quality, but also about roles, responsibilities, learning, openness to constructive criticism, and loss of some of the mystique that has long inhibited truly high-quality development.

Quality results require a different approach—use the tools and concepts in a successful manner and better results are achieved.

hotline on OBJECT-ORIENTED technology

Robert Shelton, Editor

THE HOTLINE ON OBJECT-ORIENTED TECHNOLOGY (ISSN #1044-4319) is published monthly by SIGS Publications, Inc., 588 Broadway, NY, NY 10012. (212)274-0640. Mailed First Class. Subscription rate — one year (12 issues) $249, Foreign and Canada $279. Single copy $25.

POSTMASTER: Send address changes & subscription orders to HOTLINE, Subscriber Services, P.O. Box 3000, Dept HOT, Denville, NJ 07834.

Submit editorial correspondence to Robert Shelton, 1850 Union Street, Suite 1548, San Francisco, CA 94123 voice: (415) 928-5842; fax: (415) 928-3036.

SIGS PUBLICATIONS

Publishers of HOTLINE ON OBJECT-ORIENTED TECHNOLOGY, JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, THE X JOURNAL, C++ REPORT, THE SMALLTALK REPORT, and THE INTERNATIONAL OOP DIRECTORY.

ming, and other automated software development support. These new tools and methodologies have enabled programmers to generate with increasing rapidity an unprecedented amount of code. The result is a backlog of software that cannot be adequately tested with today's debugging tools and manual testing methodologies. More than ever, there is a profound need for automated testing methods that help increase the attention paid to software quality.

Moreover, the emergence of graphical user interfaces (GUIs) and cross-platform software applications has created the need for a new generation of graphical-based test tools that operate independent of the platform, operating system and language. By remaining outside the system under test, the test tool avoids obsolescence and a single tool can systematically test and evaluate all current and future software applications.

SOFTWARE DEVELOPMENT: BLACK MAGIC OR SCIENCE?

There exists in the American software industry a mythical six-stage software development process (the "waterfall"). This schedule is predicated on the belief that:

1. System design requirements will remain frozen over the entire development period.
2. Development resources will remain stable.
3. Software will be tested for up to half the period using state-of-the-art debugging tools.
4. The project schedule is adequate for all stages of the process.

At the completion of this development process, the myth holds, the software will be bug-free and ready for market.

In the real world, however, software testing is generally regarded as the step-child of the software development process, rarely receiving the attention it deserves. Based on the assumption that software can never be perfect, software management has come to accept the notion that incomplete or flawed software is acceptable. This dangerous line of reasoning is often used by management to compress testing cycles to ensure a timely delivery to market. And management typically reinforces the acceptance of flawed products by rewarding software developers for their creativity and ingenuity, not their quality. The result is often defective software and a continual devaluation of the software testing process.

Too often software testing is relegated to second-class status in the development process because the system design is always more difficult and complex than originally planned. Additional code and significant design modifications are frequently required. Because the ship date must remain constant, delays in development frequently diminish the software testing phase. The real-world software development process reinforces the all-too-commonly held assumption that software testing is inconsequential, an impediment to the development and financial goals of the software vendor.

THE DEMISE OF QUALITY

The demands of a crowded and competitive marketplace require software vendors to deliver product in a timely fashion. Because delays in ship dates can be devastating to a company's market share, management has come to accept the short-term solution of shipping incomplete and buggy software.

Beta testing and early adopter programs are often used as a substitute for thorough system testing, creating the potential for defects to go undetected. While beta testing was originally conceived as a method for evaluating the customer's software preferences, it now has grown into a bug detection methodology. The problem, however, is that a corporate beta tester rarely has the time to significantly evaluate the product; when bugs are detected, the user usually gets frustrated and discontinues evaluating the product without forwarding the relevant information to the test engineer.

Another explanation for the prevalence of defective software is a complacent and undemanding software consumer. For whatever reason, the public perpetuates the existence of software bugs by continually purchasing defective and incomplete products and by tolerating vendors who expect customers to spend hours or days strip-testing the problem on their own machines as though they have no real work to do.

CURRENT QUALITY SOLUTIONS

Intrusive Software Tools

Software-only test tools, which have gained in popularity with the advent of windows-based GUIs, reside on the same CPU with the application under test. Priced between $300 and $6,000, these tools help automate the regression testing process at the user interface level because they interact with the application under test via the windows application programming interface (API).

Automated regression testing is a method whereby a test suite (a group of test cases) is recorded and then executed on every software build in the development cycle. This method helps eliminate the tedious process of manually retesting every software build and ensures that any code repairs or enhancements made as a result of test findings do not cause any additional software bugs or side effects. Moreover, software-only tools manage the capture and playback of test scripts, allowing test engineers to reconstruct exactly what caused an unexpected failure.

The problem with software-only test tools, according to Heisenberg's principle, is that the introduction of an outside element affects the thing under test. By sharing CPU cycle time (and other resources such as stacks and interrupts) with the software under test, the performance of the machine is altered, thereby rendering inexact test results.

Software-only tools are operating-system and platform-dependent. As a result, test engineers are required to purchase and learn a different test tool for every operating system and platform that supports their software application. While test engineers can port a specific test tool to a different platform, this process typically diminishes the test tool's effectiveness and introduces the potential of bugs into the ported tool. This obsolescence and non-adaptability factor results in increased time spent learning new tools and less time spent evaluating software.

continued on page 10

FYI

Excerpts from leading industry publications on aspects of object technology

DATABASES

. . . What's more, the object-oriented paradigm—through the notions of encapsulation and inheritance (reuse)—is designed to reduce the difficulty of developing and evolving complex software systems of designs. This is precisely the goal that drove the data management technology from file systems to relational database systems. An object-oriented data model inherently satisfies the objective of reducing the difficulty of designing and evolving very large and complex databases. Encapsulation and inheritance are a key to the search for further productivity enhancement in database application development. . . .

Unifying the relational and object models, Won Kim, DATAMATION, 11/1/92

If you don't know how—or if—your company will be moving to object-oriented technology, take pride in this, most relational database vendors haven't fully decided how to handle objects, either. In fact, vendors differ on what constitutes an object database management system (ODBMS) and how object-oriented and other unstructured data types—voice, graphics, video images and objects created through object-oriented programming (OOP) languages—will coexist with character-based data stored in relational formats. . . .

. . . Vendors are also building new features into their database engines to allow users to query both object-oriented and relational data either in some extended version of SQL or in one of two OOP languages, C++ or Smalltalk. [Codeveloper of the DB2 relational DBMS while at IBM, and now vice president of database technology at Oracle Corp, Jnan Dash] says that the next published specification for SQL, expected next year from the American National Standards Institute (ANSI), will includedstandards specifying how vendors should implement these extensions. . . .

RDBMS vendors face an object future, Mike Ricciuti, DATAMATION, 11/1/9

. . . Is the decomposition of the Open OODB system into modules arbitrary, or will other efforts to build a system with similar functionality result in a similar factoring? It is too early to report that such experiments necessarily result in similar factorings, but the Open OODB's factoring into modules is very similar to the application integration framework being developed by the industrial consortium Object Management Group. . . Thus, the OMG and the Open OODB architectures are almost isomorphic. It is interesting that one is viewed as an application integration framework architecture and the other as an OODB architecture. . . .

Architecture of an open object-oriented database management system,
David L. Wells, Jose/ A. Blakeley, and Craig W. Thompson, COMPUTER, 10/92

. . . The power of objects is in their robustness, extensibility, flexibility, and modularity. Actually I wish engineers did not have to know or care about objects. Except as interesting metaphors, they are not useful to anyone but computer professionals. But we are not yet able to reach that level of information hiding. If you are selecting an engineering database management system today, it probably should be object-oriented—and if it isn't, you should know why not.

What's the big deal about objects?, Joel N. Orr, COMPUTER-AIDED ENGINEERING, 11/92

LANGUAGES

As a C programmer moving to C++, you face a double challenge that can put you into an endless loop of misunderstanding. C++, as an extension of a procedural language, lets you use the principles of object-oriented programming (OOP) but does not teach them to you. Yet getting up to speed in any new language requires memorizing a lot of the language syntax. And, learning syntax by brute-force memorization is inordinately time consuming if you lack a higher level view of its purpose, such as, for C++, implementing OOP principles. One way to break out of this loop and teach yourself the missing chunk of C++ is to aim for a detailed understanding of how you use C++ to implement the core OOP concept, inheritance. . . .

Build a strong foundation to program in C++, John C. Napier, EDN, 10/29/92

work. object-Menu currently allows the C++ developer to rapidly integrate a state-of-the-art graphical user interface environment into a DOS-based application. object-Menu version 2 provides a seamless port of its advanced DOS features into Microsoft Windows. object-Menu is priced at $369. Source is available for an additional $529

Island Systems, 7 Mountain Road, Burlington, MA 01803, 617.273.0421, fax: 617.270.4437

Objectivity Inc.

Objectivity/DB Version 2.0 introduced new capabilities for distributed object-oriented applications, Objectivity/SQL++ for ad-hoc query using SQL, and new application lifecycle features that help administer distributed databases and update deployed applications. Objectivity/DB is currently available on UNIX workstations from DEC, HP, IBM, Silicon Graphics, and Sun, as well as on DEC's VMS operating system on VAX computers.

Objectivity Inc., 800 El Camino Real, Menlo Park, CA 94025, 415. 688.8000

Mercury Interactive Corporation

TestRunner for Windows NT lets developers automatically verify and validate all of the features and functions of Windows NT software applications utilizing output synchronization (patent pending) and text recognition technologies to monitor events on the computer screen—processing all I/O in real time. Per seat cost for a typical installation is approximately $30,000.

Mercury Interactive Corporation, 3333 Octavius Drive, Santa Clara, CA 95054, 408.987.0100, fax: 408.982.0149

ICL Inc.

ICL announced Dialogue Management System (DMS), a client development environment for online transaction processing that integrates multiple existing TP systems across an enterprise without reengineering. Users of DMS can access the enterprise's varied TP systems as if they were a single integrated service with an easy-to-use front end. Currently on limited release for pilot development in partnership with ICL, general availability is planned for December 1992.

ICL Inc., Press Office, 9801 Muirlands Blvd., P.O. Box 19593, Irvine, CA 92713-9593, 714.458.7282, fax: 714.458.6257

Virtual Technologies Inc.

Virtual Technologies announced new ports for the SENTINEL Debugging Environment: HP 9000 Model 7xx, IBM RS/6000 model 2xx /3xx /4xx /5xx /7xx/9xx, NCR UNIX, and SCO UNIX and Open Desk Top (ODT). SENTINEL is designed to assist C/C++ programmers in locating and resolving hidden bugs with the use of dynamic memory, as well as assisting developers in determining the cause of memory leaks.The SENTINEL debugging environment is priced from $195 to $795 depending on the platform architecture. Substantial discounts apply for multiple unit purchase.

Virtual Technologies Inc., 46030 Manekin Plaza, Suite 160, Dulles, VA 20166, 703.430.9247, fax: 703.450.4560

Dashboard Software

Dashboard Software released TrackDeck, a new programmer's utility for Windows and OS/2 developers. TrackDeck is a software dashboard that allows you to examine any variable from your code and track its value as your program executes at its normal speed. In addition to finding program bugs, TrackDeck's control panel lets you set up permanent displays of crucial parameters. TrackDeck works with any language that can access DLLs. There are special interface components for C/C++. TrackDeck for Windows is now selling at a promotional price of $129. TrackDeck OS/2 license agreements are available.

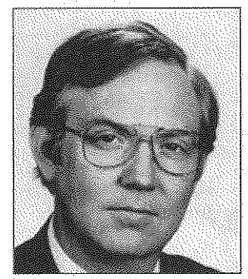
Dashboard Software, 4 Louis Avenue, Monsey, NY 10952, 914.352.8071, fax: 914.352.8071

McCabe & Associates

McCabe & Associates announced the release of BattlePlan, a forward engineering addition to the McCabe Tools Set, which provides forward engineering within the reverse engineering environment. The tool gives software developers instantaneous testing, verification, integration, and application of the McCabe methodology in the design phase—showing discrepancies between what you said you wanted to do and what the code you generated actually does. BattlePlan works with all the languages and dialects currently supported by McCabe Tools, including C, C++, Ada, and FORTRAN.

McCabe & Associates, 5501 Twin Knolls Road, Suite 111, Columbia, MD 21045, 301.596.3080
Baltimore 410.995.1075, 800.638.6316, fax: 410.995.1528

MIS radar detects objects for the first time



Tom Love

Objects appeared on the radar screens of leading-edge MIS directors for the first time in 1992. A little awareness by MIS directors can mean a huge increase in a tiny (few hundred million dollar) market like the O-O market.

For the first time, several major corporations kicked off object-oriented projects to replace traditional batch data processing systems. Most remarkably, more than one such project is being done in Smalltalk. The batch systems are being redesigned as distributed systems with the data residing on one or more servers or host computers and the "application code" residing on workstations. Building distributed systems requires objects.*

In 1992, objects gained penetration in three market segments—network management systems, the oil and gas industry, and the securities market. For any given new project beginning in one of these markets, there is a better than 50% chance that objects will be used. I am personally aware of more than 10 major development projects under way in each of these market segments.

The consolidation of the industry predicted last year (see my article in HOTLINE 3[4]) has continued throughout the year. Liant acquired CNS's C++/Views product, Borland acquired Brief for C++, Symantec acquired Gain Wong from Borland and Whitewater Group from its investors, ParcPlace has acquired Infoware from Ensemble Software and the C++ tools development group from Solbourne, AMS acquired a stake in KSC, Gemini Consulting acquired a stake in ParcPlace, Computer Associates acquired Glockenspiel, and Enfin was acquired by Easel Corporation.

Even James Martin has begun to invest in object technology companies (Versant and IntelliCorp).

Momenta also managed to ship an aggressive product built using Smalltalk but later had to ship the company's keys back to the venture capitalists when disappointing hardware performance was coupled with a lackluster response from the market.

Meanwhile, Taligent was formed as an independent joint venture between Apple Computer Corp. and IBM. Taligent's charter is to develop a completely object-oriented environment from the operating system up.

Sun has embarked upon a major project to produce an object-oriented distributed environment called Project DOE. To support this major new development, the company has been acquiring O-O talent at a prodigious rate. Its most notable acquisition is Bud Tribble, a vested founder of NeXT Computer Corporation. Only a few months before initial product was expected, Sun announced a technical joint venture with HP to build a distributed operating system. One can only guess that as the project progressed, a more capable O/S was found to be extremely desirable. This unquestionably major undertaking will consume Sun for at least the balance of the decade .

On the product front, Microsoft now holds the record for having shipped the "biggest" product—a C++ compiler and class library with 4,000 pages of documentation. One person is rumored to have read it all, but she hasn't slept in months.

One of the year's most innovative products came from the object masters at HP in Loveland, CO. In my opinion, the VEE (visual engineering environment) product is a sleeper. This product is actually a higher level environment within which arbitrary objects can be assembled to form assemblies of objects. It is fast, transportable, and very capable. See the October issue of HP JOURNAL for three articles describing this product and some experiences building it.[1,2,3]

NeXT Computer is about to become the most recent addition to the list of object-oriented software companies as it begins its challenging transformation from a high-end hardware supplier to a software company trying to build vendor-independent software development environments. Yet as one wanders around the financial community, one sees more and more trucks lining up at the loading dock with NeXT workstation boxes.

One of the more significant happenings this year has been the emergence of Smalltalk as an application development environment for commercial application developers. American Airlines, for example, has deployed a commercial system to manage the resources required for all flights worldwide. This high-reliability, high availability distributed system was programmed in Smalltalk and is considered a major success.

1992 was also the year that Smalltalk companies got "professional management." As one with some direct experience in this area, I am very skeptical of any software company leader who cannot use the products made by that company. "General managers" can solve some problems of an emerging company but, ultimately, deep knowledge will be required to make the tough decisions required to grow and prosper. I sincerely hope these new CEOs can prove me wrong.

The other challenge facing new professional managers of Smalltalk companies is that MIS directors can be very demand-

* See Chapter 5 in my new book, OBJECT LESSONS, to be published soon by SIGS Publications.

ing to do business with. They demand services and insist upon delivering new products on or about the published schedules. As they evaluate Smalltalk, they see a lot missing. The challenge for the next couple of years will be to rapidly add capability without losing focus. Development environment companies should build strong development environments and kernel classes for their language. Their business partners should build CASE tools, development methods, specialized class libraries, database systems, report writers, and communication software. The goal: focus, deliver, and hope to dominate. Trying to provide one-stop shopping is not possible.

Just as Smalltalk has begun to creep into mainstream businesses, the harsh, cruel realities of using C++ as an application development language have been felt in company after company. While C++ can be used as an object-oriented language, it typically is not. Rather it is used as a more complex C with esoteric new features that someday must be understood. NeXT and Apple continue to plug the virtues of a dynamic object-oriented environment but they don't get heard above the shrill voices of professional marketeers selling compilers in the ads of street-corner magazines.

(Here's a trivia question: can you name the CEO of a company that exhibited at OOPSLA '86 who is still that company's CEO? To my knowledge there are none, though some former CEOs remain with their companies in another role, often as Chairman.)

1992 saw serious commercial products being deployed with OODBMs for the first time. The most compelling example is the Air Fone system of Hughes Network Systems, which involves over 100 users connected to the Versant OODBMS. Another is Domestic Automation, which has delivered a commercial network management system product based upon Object Design's ObjectStore product. There are a number of further examples from these vendors as well as others.

The O-O methodology business has been awash with new books touting new paper and pencil graphical notations that produce "better" O-O designs. Most of these "diagram touters" have less than two years' experience in the O-O business. Serious O-O designers read, listen, then pull out their index cards and proceed to analyze and design with the help of a couple of experts. These same designers look to people like Booch, Rumbaugh, Jacobson, and Embley (and HP), hoping they can provide real leverage and real tools that will help.

A rather startling change has been in the paychecks of highly competent O-O designers and developers. Some have doubled; a few have tripled in the last year. Companies are beginning to recognize that someone who really knows existing object-oriented libraries and tools can be worth more than five greenhorns. For this time-to-market advantage, they are willing to pay handsomely. I have seen individual Smalltalk programmers working for $2,000 per day on long-term contracts and Objective-C programmers making a salary of $200,000 per year. And this trend will accelerate.

Training companies can't find enough qualified instructors to satisfy the demand; this shortage will worsen substantially next year. My advice to large companies is to book your training courses for next year now. Otherwise you will have to wait or settle for brand-new instructors who are only a few steps ahead of their class.

What can we predict about this industry for the coming year? Mainframe computer companies and software companies will start noticing objects and buying their way in the door. Some of these companies are loaded with cash and have an imminent need to change their strategy as the mainframe business evaporates during the coming decade.

> *In the coming year mainframe computer companies and software companies will start noticing objects and buying their way in the door.*

As the big boys enter the game, expect it to be the year that irrational hopes meet the harsh reality of building commercially distributed systems with objects. Adopting objects does not create instantaneous nirvana nor spectacular productivity. Yet you will achieve neither by failing to adopt objects. Experience says that much hard work and dedication are required to learn how to build distributed object-oriented systems. You should think about transforming your organization over a decade, not a year.

2003 is sooner than you think.

P.S. No company managed to ship a class library with one test case for every method again this year. My challenge remains in effect!

References

1. Hunt, W. L. and D. C. Beethe. A visual engineering environment for test software development, HP JOURNAL 43(5):72–77, 1992.
2. Hunt, W. L. Developing an advanced user interface for VEE, HP JOURNAL 43(5):78–83, 1992.
3. Beethe, D. D. HP VEE: a dataflow architecture, HP JOURNAL 43(5): 84–88, 1992.

Dr. Tom Love is President of OrgWare, Inc. in Roxbury, CT and an independent software consultant specializing in object technology. In 1983, he founded Stepstone, the first company to deliver an O-O development environment and libraries. Dr. Love is author of the book OBJECT LESSONS, available this spring from SIGS Books, and host of the training video DESIGN MASTERS, also available from SIGS. He can be reached at 203.350.4331.

Product Announcements

Product Announcements is a service to our readers. It is neither a recommendation nor an endorsement of any product discussed.

Pacific HiTech, Inc.

Pacific HiTech, Inc. announced plans to regularly release updates to its Info-Mac CD-ROM disk. The first edition of the disk, released in September, contains 3,384 shareware and freeware Macintosh programs and files. The disk programs came from the Info-Mac archive on Internet. Since so many new programs appear every week on Internet, Pacific HiTech is planning to update the disk four times a year. Info-Mac CD-ROM, priced at $39.95, is available directly from Pacific HiTech.

Pacific HiTech, Inc., 4760 Highland Drive, Suite 204, Salt LakeCity, UT 84117-5009, 801.278.2042

Quinn-Curtis

Quinn-Curtis announced the Windows Charting Tools for Microsoft C/C++ and Borland C+ programmers, a collection of general-purpose graphics and user-interface routines that solve the most common charting problems encountered in scientific, engineering, and common business charting applications. The product has been specifically written for the Microsoft Windows 3.1 programming environment. Standard Package, $400; with complete source code to the Quinn-Curtis Charting DLL, $800.

Quinn-Curtis, 35 Highland Circle, Needham, MA 02194, 617.449.6155, fax: 617.449.6109

SunSoft

SunSoft introduced Solaris 2.1, its distributed computing environment. The Solaris 2.1 computing environment incorporates more than 1,000 product improvements, delivering up to a 40% increase in areas of network performance and up to a 50% increase in user interaction performance over Solaris 2.0. The Solaris 2.1 software includes full symmetric multiprocessing and multithreading capabilities, in addition to major enhancements in the graphics area, providing users with 2D-3D capabilities. The Solaris 2.1 environment for the desktop is priced at a suggested retail price of $795 in single quantity. Volume discounts to system manufacturers are available.

Sunsoft, 2550 Garcia Avenue, Mountain View, CA 94043 415.336.0678

ImageSoft Inc.

ImageSoft Inc. announced it is shipping Object/Designer its extensible C++, C, and Pascal generator for Windows. Object/Designer 2.0 includes the ability to generate custom source code for custom controls and user code regeneration. ImageSoft also announced the availability of the Microsoft C/C++ version of ImagingObjects 2.0, its C++ imaging toolkit. ImagingObjects 2.0 enables developers to easily write imaging applications. A Windows 3.1 interface kit is included with ImagingObjects 2.0.

ImageSoft Inc., 2 Haven Avenue, Port Washington, NY 11050, 516.767.2233, 516.767.9067

Scientific and Engineering Software, Inc.

Scientific and Engineering Software, Inc. (SES) is expanding into the object-oriented application software development marketplace with SES/objectbench, a UNIX-based, object-oriented analysis (OOA) toolset for developers and programmers using the C and C++ languages. SES/objectbench—based on the Shlaer-Mellor OOA methodology—offers on-screen animation and dynamic checking of models. A "VCR-like" interface allows users to watch models execute. The product is priced at $14,500 per concurrent user, which includes the animated simulation feature. The graphic capture package feature is offered for an additional $4,900 per concurrent user.

Scientific and Engineering Software, Inc., 4301 Westbank Drive, Building A, Austin, TX 78746-6564, 512.328.5544, 512.327.6646

ConVal Software

ConVal Software introduced ToolDriver, a software management environment for MS-DOS with cross reference and naming convention changing capabilities. It supports software written with any combination of Oracle, C, C++, Pascal, COBOL, FORTRAN, batch files, and other languages. It also works with text files containing project documentation. ToolDriver's software management environment has a window interface complete with menus, mouse support, and online help. ToolDriver is priced at $59.95.

ConVal Software, 11607 E. Butter Creek Road, Moorpark, CA 93021, 805.529.6847

Island Systems

Island Systems announced cross-platform support for the object-Menu DOS graphics application frame-

Industry Briefs

Trinzic Corporation, the company formed by the merger of AICorp and Aion, announced an expanded marketing agreement with **Stone & Webster Advanced Systems Development Services, Inc.** Under the agreement, Stone & Webster will market, license, and support Trinzic's Aion Development System (AionDS) in conjunction with STONE rule, a sophisticated software tool that allows for the use of knowledge base systems during computer-aided design sessions.

Objectivity Inc. and **MICRAM Microelectronic GmbH and Co. KG** announced a distribution relationship. Under the terms of the agreement, MICRAM will distribute Objectivity's Objectivity/DB object database management system in Germany. MICRAM will handle sales, marketing, and customer support in Germany. In addition, Objectivity and MICRAM will work together on joint projects with customers.

Objectivity Inc. announced it would support **SunSoft**'s Project DOE (Distributed Objects Everywhere) by delivering the Objectivity/DB object database management system with an interface to the SunSoft DOMF (Distributed Object Management Facility), SunSoft's implementation of the Object Management Group's Common Object Request Broker Architecture (CORBA). Objectivity and SunSoft also agreed to jointly develop and publish an open specification to provide other ODBMS vendors with a standard interface between DOMF and ODBMSs.

Objectivity Inc. announced that its object database management system, Objectivity/DB, was selected by Adra Systems Inc. of Lowell, MA, for use in the development of its next-generation, mechanical computer-aided design (MCAD) and product management systems.

Objectivity Inc. and **Spatial Technology Inc.** announced that Objectivity will provide ACIS/DB, an integration between Spatial Technology's ACIS geometric modeler and the Objectivity/DB object database management system.

Virtual Technologies appointed **Engineering Software Ltd**, based in Manchester, England, to be its new European distributor for its SENTINEL debugging environment.

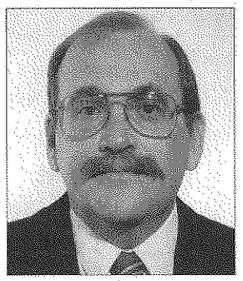
Object Technology International announced the integration of Visualworks, ParcPlace's recently announced product, with ENVY/Developer. Availability and pricing for this integration has yet to be finalized. Object Technology and Servio Corporation also announced a cooperative relationship to support Smalltalk application development and delivery.

Visual Edge Software, Ltd. announced that its interface development tool, UIM/X 2.0, will be available for use with Object Design, Inc.'s ObjectStore Release1.2.

Versant Object Technology Corporation announced that it signed agreements to integrate the VERSANT Object Database Management System (ODBMS) with five leading object-oriented development tools. Agreements were made with IntelliCorp (PROKAPPA); Lucid, Inc. (Lucid's Energize Programming System); STEP (STEP Toolkit); Protosoft (Paradigm Plus); and Persistence Software Inc. (Persistence).

KAPRE Software, Inc. announced the successful completion of its second refinancing for the development of KAPRE's Object-Oriented Application Software and C++ Application Development Environment.

In Pursuit of Object Engineering



Richard T. Dué

"Where do objects come from? . . . Heaven!" is how Dr. Adele Goldberg, Chairman of ParcPlace Systems, summarized the "just do it" mentality of many of today's object technology practitioners during her presentation at OOPSLA '92.

According to Dr. Goldberg, the deficiencies of current object technology projects are related to the lack of object paradigm training and education; the lack of effective OOP, OOA, and OOD systems development and project management methodologies; the lack of systems development metrics; and the lack of effective reuse policies.

Objects don't come from heaven but from dynamic, reusable simulation models of the enterprise. Strategic understanding of the enterprise is key to reuse in-the-large (at the business process level) instead of reuse in-the-small (code scraps level).

This article presents three key concepts necessary to the implementation of reuse in-the-large: the layered class library, design by contract, and replicable modeling.

The integrated application of these three concepts in a new planning, analysis, and design methodology may be key to the development of a strategic, object-engineering approach to object technology.

The development and implementation of object technology seems to be following the same bottom-up path as the development and implementation of software and information engineering (see Table 1). It has taken 25 years for software engineers to learn that developing good structured code requires good structured design dependent upon good structured analysis dependent upon a good strategic plan linked to the goals of the organization. Hopefully, OT practitioners will apply the lessons of the software and information engineering learning curve, instead of try to develop bottom-up O-O programing, design, and analysis techniques and methods in the absence of a strategic, disciplined understanding of the enterprise. OT practitioners hopefully also will learn that the time-consuming, resource-intensive, top-down, static-enterprise models developed by information engineers must be replaced by dynamic, iterative models representing continual integration of the enterprise's applications.

Object technology is not just an incremental improvement to programming languages or to analysis and design techniques. It is a fundamentally new way of looking at the planning, development, and maintenance of information systems. Instead of the process and data-oriented concerns of software and information engineering, object engineering is concerned with the reuse of models that actively simulate the enterprise. Reuse at the enterprise level allows users to analyze, design, and implement systems that are direct representations of the enterprise. Dr. Brian Henderson-Sellers forecasts that 50% of systems will be developed in O-O by the year 2000. He points out that we must first teach the paradigm, not an O-O language, because there is, as yet, no perfect O-O language.[2] In fact, by shifting focus to reuse in-the-large of previously developed models from creating code, object engineering may render the requirement

Software/Information Engineering	Object Technology
Structured Programming Dijkstra, Parnas (1960s–1970s) Four basic programming structures (sequence, do while, if then else, case)	Object-Oriented Programming Some preliminary work (Coad's object patterns, Smalltalk's model view controller, but no generally accepted standards
Structured Design Yourdon, etc (1970s) Inputs, Central transform outputs, structure charts, HIPO diagrams	Object-Oriented Design Scenarios, use cases, CRC, numerous notations, but no disciplined methods
Structured Analysis DeMarco, etc. (1970s) Data flow diagraming, functional decomposition	Object-Oriented Analysis Numerous notations but no disciplined methods. Some preliminary work by Henderson-Sellers and Edwards[1] and ParcPlace Systems' object behavior analysis
Information Engineering Enterprise Modeling Martin, Finkelstein (1980s)	Object Engineering Some preliminary work by Jacobson and Henderson-Sellers

Table 1. A comparison of software/information engineering and object technology.

for a perfect language superfluous to most application developers. Instead of our current concern with developing OOP, OOD, and OOA techniques, object engineering will allow us to concentrate on the real payoff issue: the stepwise development of reusable business models.

To use the full power of the object paradigm we have to overcome a significant resistance to change. Discussing the difficulties encountered in trying to convert traditionally trained information systems developers to the object paradigm, Charles Kahn writes:

> Object-oriented programming has no concept of a function being called to perform some algorithm and produce some result. To think of a message passing facility as simply a function call, however, would be to look once again at the problem in a functional manner and would defeat the goal of an object-oriented approach to the problem. This problem of a message being implemented as a function is one of the major arguments against object-oriented extensions to more traditional, procedural languages (e.g., C++ extensions to C). It seems that most professional programmers often revert to a functional viewpoint when implementing a program if not forced by the language to work in an object-oriented fashion.[3]

The first key concept for object engineering is the layered approach to class libraries, which views classes from four levels of abstraction (see Table 2). The first level consists of basic, or atomic, classes, where O-O programmers develop the lowest level system building blocks. This level may be subdivided into further layers or clusters of classes that provide basic services for string handling, graphical user interface, mathematical functions, database, communications, etc. This level probably contains 5,000 or more members (an estimate based on the size of existing Eiffel, C++, and Smalltalk libraries), which should be developed by OOP language or class library vendors, and provides the essential infrastructure of the object technology approach.

The second layer consists of business process classes, which are the fundamental, stable building blocks of all applications. Data is not necessarily the stable component of the enterprise, especially as enterprises continually reinvent themselves, merge, downsize, and divest. The stable parts of the enterprise are the basic business processes of purchasing, reporting, control, security, etc. These classes are specified by business systems analysts by combining the basic Atomic classes. Little or no code development should be performed at this level. Surprisingly, based on preliminary estimates, this layer probably only contains 20 to 30 business process models. These fundamental models include processes like mediation (one object requesting a service from another object via a third object), transaction (one object requesting a service directly from another object), transformation (conversion of a service into another form), edit (verification of a service against a standard), etc. If there is a need for additional atomic classes to specify a business process, business systems analysts must negotiate with atomic class developers to provide the new atomic classes. Business systems analysts should not be allowed to program new atomic classes.

The third layer consists of management or application classes. Developed by users, these classes are assemblies of business processes. Only minimal coding (just enough to tie together existing classes) should be undertaken at this level. If business process classes do not exist in the business process layer for application modelers to use, the user must negotiate with the business process modeler to provide the required class.

The fourth layer of the model is the enterprise engine level. At this layer, object engineers model the interaction of the application classes to provide a dynamic simulation of the enterprise. These dynamic, real-world simulations will eventually evolve into the accounting, communications, and MIS reporting systems of the organization.

The layered approach offers an opportunity to leverage and reuse work undertaken at the lower levels of the class library. People working at their own levels have no need to understand the internal workings of other levels. As Jeff Sutherland, a trainer at Semaphore, describes it:

> No matter how long you study cells in the human body (the atomic layer), you will never understand the English language (the application layer)!

Layer 4. ENTERPRISE ENGINE—object engineers (A distributed expert system—inference engine, rule base, database) Gradual merging of models and applications, collaborative effort throughout organizations, depends on standard classes and structures. Not just a model—an active engine that continues to evolve along with the company
Layer 3. APPLICATIONS—prototypers, users Solves a business problem, can mix and match models, independent of model details, very little new code, software by assembly, built through rapid prototyping
Layer 2. MODELS—model builders, business systems analysts Comparable to PC boards, handles standard functions (purchasing cycle, customer interactions), maximizes reuse of every class, corporate process more stable than applications
Layer 1. CLASSES—class constructors Libraries, software IC's, standard components, reusable functionality, common business objects, standard packaging, multiple vendors

Table 2. The Layered Model.[4]

The Australian Object-Oriented Scene

by Brian Henderson-Sellers

You may wonder at my silence over the last few months. The major reason is that for the first half of 1992 I was based in the United States and so unable to comment on the Australian O-O scene, where object-oriented interest continues to grow more rapidly. Since returning, I (and others) have been inundated by requests for information on object-oriented techniques from organizations wishing to commence object-oriented developments, and from people already working in this area who wish to update their knowledge of this rapidly changing technology. It is encouraging to see the increasing acceptance of O-O as part of mainstream information systems, computer science, and software engineering, rather than as a curiosity demanding its own conferences and its own alienated devotees. Speakers on O-O are now slotted into timetables at conferences not specifically focused on O-O. Such acceptance is surely what we are all seeking!

In addition to increasingly frequent appearances at more traditional organizations and conferences, there are still, of course, specialist O-O meetings and conferences. We have seen "The Object-Oriented Symposium," SPOOK and TOOLS in 1992, and are looking forward to the first Object World to be held on this continent in 1993.

Australian object-oriented special interest groups go from strength to strength, with the main two, Victoria and New South Wales, trying to outdo each other! Rivalry is fun as is the close cooperation between several groups across the country. Networking, both interacademic and academic-industry, is basically good.

One of the concerns most frequently voiced by Australian industry is the current lack of good CASE tools. Those available either seem to have no Australian distributor or have such heavy computing demands that those still focused on the DOS market for their customers are denied access. Project management guidelines and metrics are also sorely needed but their prognosis is good for the relatively near future.

Also encouraging is the number of graduates with O-O skills being sought. Universities around the country are beginning to respond by introducing good O-O courses. Most of these are still at the more specialized level (final year Bachelor's and options on Master's programs); ideally students should commence training in their first term but few universities are yet able to offer that.

There is, of course, one overriding advantage to working with objects in the southern hemisphere—we are still looking forward to this year's summer!

Brian Henderson-Sellers is associate professor in the School of Information Systems at the University of New South Wales and chairman of the O-O special interest group of the NSW Branch of the Australian Computer Society. He can be contacted via brianhs@cumulus.csd.unsw.edu.au

CALENDAR

To have a meeting or conference listed, please send the dates, conference name and location, sponsor(s), and contact name and telephone number to Dylan Smith, 588 Broadway, Ste. 604, New York, NY 10012, fax: 212.274.0646.

February 1–4 and February 4–5, 1993 OOP '93 and C++ World Munich, Germany Contact: 212.274.9135	February 21–26, 1993 Software Development Spring '93 Santa Clara Convention Ctr., Santa Clara, CA. Contact: 415.905.2319	March 8–11, 1993 X World New York, NY Contact: 212.274.9135	March 8–10, 1993 WOOD '93 Snowbird Conference Center, Snowbird, UT Contact: 414.789.5253	March 8–12, 1993 INTEROP Washington, DC 800.INTEROP
March 8–12, 1993 TOOLS Europe '93 Versailes, France Contact: +33.1.45.32.58.80	March 17–19, 1993 Uniforum '93 San Francisco, CA Contact: 800.323.5155	March 30–April 1, 1993 Object Technology '93 Cambridge, England Contact: +44.491.410222	April 19–23, 1993 Object Expo Hilton Towers New York, NY Contact: 212.274.9135	May 3–7, 1993 DP Expo San Francisco, CA Contact: 415.966.8440u

formation models. One alternative use of an information model is to "recover" a database design, i.e., reverse engineering. The basic premise is that by documenting and normalizing an existing system you will understand the nature of your legacy systems. This understanding then can be used in forward engineering a new and improved system. The technique described in the book is complete and rigorous but, as Mr. Bruce points out, the final result is based on past mistakes augmented by analysts' assumptions about the intent of the original designers.

One practical issue is to support information models with CASE tools and repositories. The author provides an overview and sample output of commercial tools that support the IDEF1X notation and includes vendor contact information. A brief description of the ANSI, ISO, and IBM repositories is presented along with a discussion of their common traits and distinguishing features. A more comprehensive description of IBM's repository modeling language is included in the appendix.

A second practical issue is how to conduct and what to expect from your modeling sessions. This material is presented in an appendix that summarizes the rules and steps of modeling sessions—sort of a one-minute guide to the fields of interviewing and time management. An elaborate case study is used to demonstrate how an information model can be developed, reviewed, and evolved during several modeling sessions. The case study is presented as an annotated transcript of a series of modeling sessions and the resulting information model. This rather unusual format is very effective in conveying the tenor and pace of an information model developed with a supportive user. The natural ebb and flow of analysis become clear as you read the transcript and match it against the evolving data model.

The fourth and final section describes Robert Brown's evolution of IDEF1X into the object modeling language DMT/2.* DMT/2 extends IDEF1X by including methods in the current diagramming notation and adding a message pattern view of the model. These enhancements are similar to the work by Rumbaugh, Shlaer/Mellor, and others who are extending traditional structured techniques. The use of a semiformal specification language to describe methods and rules sets DMT/2 apart from these other approaches. The language is declarative in nature and intended to be executable. If realized, this opens significant possibilities for developing models that can drive simulations, act as prototypes, or be included in a multimedia presentation. A second major difference is DMT/2's use of condition/action rules. These are written in the specification language as a query over object property values that invoke a method if true. The definition of rules appears to be independent of the definition of methods or object/entity-types.

I recommend this book to anyone involved with information. A novice would be spared many hours of frustration. As an experienced modeler I found numerous insights, tips, and warnings. A manager or instructor will find an excellent discussion of database design. The section on business rules and normalization alone justify the price. Finally, the forward look toward object modeling gives the data modeler a bridge to object technology.

Michael Fuller is an information engineering consultant and has developed large-scale, distributed applications with a variety of technologies including Eiffel. He can be reached at 415.928.7067.

* Reviewer's note: be aware that this material is now three years old and Brown has (apparently) made numerous enhancements and refinements.

The second key concept for the object engineering approach is design by contract. Each class of objects in each layer has a five-part contract describing the class's protocol. Haim Kilov, an object modeler at Bellcore, describes this contract as follows:

> **A contract may be visualized as a [service request] protected by constraints, i.e., surrounded by a precondition (specification of the conditions that must exist before the class can be invoked), a postcondition (a specification of the conditions that will exist after the class is invoked), and existing within an environment (context) characterized by an invariant (statements specifying what must always be true of business information outside of any operation).[5]**

In addition, the contract may also contain exception clauses (specifications of conditions requiring exceptions to standard behavior) and, perhaps, methods clauses (specifications of the behavior owned by the class). While encapsulation should hide the implementation of a class, human modelers apparently still want to know the methods of a class.

For example, consider a class called personnel management. One of its protocols is the ability to add an employee to an organization. The contract attached to this class would include the preconditions (budget exists, management approval exists, employee exists, etc.), the postcondition (one new employee will be added to the enterprise), the invariant (no supervisor has more than seven directly reporting employees; all employees have a supervisor), the exception (the president does not have a supervisor), and the optional method (specification of the add-an-employee process).

Design by contract is the concept of designing systems by developing contractual specifications that will be used at every level of the layered class library to specify what behaviors are required. These contracts may be written in natural language, O-O programming language (e.g., Eiffel), or, more likely, as formal specifications (e.g., predicate calculus or conceptual graphs). At a recent presentation at CASE WORLD in Hamburg, Germany, Dr. Roger Pressman stated that the formal specification of systems (especially human-critical systems like avionics, air traffic control, or intensive care medical systems software) will be accepted practice by the year 2000.

Contractual specifications and services will be matched among objects by traders (e.g., OMG's CORBA [Common Object Request Broker Architecture]). It is possible that classes will actively advertise their services to potential users through these traders.

The third key concept for object engineering is the use of replicable modeling techniques, which should lead to the development of rigorous, normalized models of the problem space. Promising approaches to replicable models include entity trace diagrams,[6] which describe the lifecycle of an object in the real world, and play scripting (modeling the enterprise as play with a number of acts and scenes).[7] Both approaches model the enterprise in terms of scenarios that describe the actual operations of the organization. The enterprise model will be developed in terms of when things happen in the organization, who performs them, where they are performed, how they are performed, what resources are used, what the purpose is, etc. The purpose of the scenario approach to enterprise modeling is to provide a common framework for communication and model comparison. Today's modeling techniques (e.g., CRC modeling scenarios from object perspective) lack the perspective of the overall enterprise. There is no notion of the enterprise's environment and no notion of sequential and concurrent operations. The dynamic-enterprise model must be composed of a number of rich semantic descriptions (e.g., data, function, behavior, resources, organization, finance, network, etc.) of the organization at various points in the enteprise's lifecycle.

> ***"The layered approach offers an opportunity to leverage and reuse work undertaken at the lower levels of the class library."***

Object engineering, with its interrelated use of layered class libraries, design by contract, and replicable modeling, promises to be an effective approach to enabling object practitioners to apply reuse in-the-large power of the object paradigm. Each object within the layered model will have its own contract; contracts will be written between layers to develop applications; and applications will be integrated into replicable play script models of the enterprise.

In future articles I will describe the resulting object engineering methodology.

References:

1. Henderson-Sellers, B. and J.M. Edwards. BOOK TWO OF OBJECT-ORIENTED KNOWLEDGE: THE WORKING OBJECT, Prentice Hall, 1993 (in press).
2. Henderson-Sellers, B. A BOOK OF O-O KNOWLEDGE, Prentice Hall, 1992.
3. Kahn, C. Object-oriented programming techniques for a traditional environment, HANDBOOK OF IS MANAGEMENT, June, 1992.
4. Taylor, D. OBJECT-ORIENTED INFORMATION SYSTEMS: PLANNING AND IMPLEMENTATION, John Wiley & Sons, 1992.
5. Kilov, H. and J. Ross. THE FRAMEWORK: A DISCIPLINED APPROACH TO ANALYSIS, Bellcore, 1992.
6. Rumbaugh, J. and M. Blaha, et al. OBJECT-ORIENTED MODELING AND DESIGN, Prentice Hall, 1991.
7. Dué, R.T. Enterprise modeling: still in pursuit, DATABASE PROGRAMMING AND DESIGN, November, 1992.

Richard T. Dué, President of Thomsen Dué and Associates Ltd., develops and presents object technology training courses. He can be reached at 403.439.4627; Internet: 70544.3665@compuserve.com.

INTRUSIVE HARDWARE TOOLS

Intrusive hardware tools consist of a video card or adapter that is inserted into the computer performing the test, along with a software testing application that resides on the same CPU as the application under test. In addition to the features found in software-only tools, intrusive hardware tools allow more precise graphics comparison testing. Graphics comparison testing, which is essential in today's world of graphical user interfaces, allows the comparison and display of expected to actual graphical test results.

> ***The need for reliable, bug-free software is no longer an issue that belongs exclusively in the software testing laboratory.***

While intrusive hardware tools add graphics comparison testing to their list of capabilities, they suffer from the same liabilities as software-only tools. These tools change test performance because the software and hardware share CPU cycles with the application under test. Additionally, these tools add a complexity of interrupt and bus resource contention, which sometimes can be an issue. As a result, test engineers never can be sure that the software that was shipped is the same as the software that was tested.

NONINTRUSIVE TOOLS

In an effort to create a new standard in software quality, software test tool vendors are beginning to develop nonintrusive software testing systems. These new computer aided software testing (CAST) systems typically consist of a workstation computer using an interface box that intercepts and converts the mouse, keyboard, and video signals from the system under test into standard protocols, which the system then can use to automate the testing process.

Unlike all other test tools, the CAST system's nonintrusive approach to software testing ensures that a single tool now can be used to test software on all popular platforms, operating systems, and language types.

A nonintrusive test tool has several advantages over its intrusive predecessors:

- **Accuracy.** Because a nonintrusive tool remains outside the system under test, it does not alter the application under test by requiring the same CPU cycles and memory needed by the system under test. As a result, software vendors can be assured that the shipped software is the same software that was tested.
- **Automation.** If a test tool causes the application under test to completely hang up, the nonintrusive test system can log the error, reboot the system, and continue testing. With an intrusive tool, if the application crashes, the test tool will crash too.
- **Non-Obsolescence.** By existing outside the system under test, a nonintrusive test tool provides the highest degree of protection against test tool obsolescence by simply adding a new interface box that can be connected to the new platform or operating system. While the software running the CAST system potentially can become obsolete, its multi-platform and multi-operating system features allow a single tool to systematically test and evaluate the broadest range of current and future software applications.

INTEGRATION

In today's world of software development, the testing phase is generally regarded as an afterthought. As a result, the test engineer, who does not have access to all requirement and design specifications, is incapable of designing test cases that will ensure comprehensive coverage. To achieve accurate test results, the testing phase must commence and proceed along with the development phase.

In an effort to integrate development and software testing stages into a single, automated process, CAST systems will be able to interface with CASE technology. The result is an automated software development system capable of automatically generating and executing test scripts derived from the data flow diagrams, structure charts and data dictionaries used to design the software itself.

With the integration of CASE and CAST, a software developer would first use CASE technology as a front-end analysis and design tool. A test case generator then would provide an automatic translation of the CASE tool's graphical notation language into test cases using well-known test design rules, which the CAST system would then automatically execute. This scenario would provide the most precise and efficient software testing methodology possible.

Integrating CASE and CAST enables the test engineer to develop testware in conjunction with software, thereby achieving the highest level of software development integration and automation.

THE QUALITY IMPERATIVE

The software industry is fast becoming a commodity market. Product differentiation has traditionally been characterized by features and performance. But now that the market has matured and many vendors are offering similar products with similar features, quality will become the characteristic that determines success in a crowded marketplace.

Achieving a higher level of quality does not happen by accident. By making long-term strategic investments in quality programs such as automated test tools, US software vendors can begin to build quality and customer satisfaction into the very fiber of the product development process.

Perhaps it is time that our industry challenge itself to produce defect-free software—before the offshore competition emerges with products that capture the lion's share of the market.

Greg Pope is the General Manager of the Test Products Group at Tiburon Systems, Inc. Mr. Pope has over 20 years' experience as a software development engineer. He can be reached at 408.293.9098.

DESIGNING QUALITY DATABASES WITH IDEF1X INFORMATION MODELS

Thomas A. Bruce
Dorset House Publishing, 1992
547 pp.

reviewed by Michael Fuller

This book could have been titled "Everything you ever wanted to know. . ." and still would have met the most stringent labeling requirements. The amount of information between the covers is difficult to absorb, even after multiple readings. Fortunately, the large type, airy layout, and Mr. Bruce's style make this an exceptionally easy book to read.

DESIGNING QUALITY DATABASES WITH IDEF1X INFORMATION MODELS is built around the premise that "an information model is developed to capture the needs and policies of a business in a uniform and unambiguous manner." This leads to a single measure of quality: "An information model is successful only when it adds value to the business." The book develops these ideas using four major sections that address the:

1. need for and perspective of information models
2. components and structure of information models
   - entity types
   - attributes
   - relationships
   - super/sub type hierarchies
   - insert/replace/delete rules
3. use and practical considerations of information models
4. evolution of information models to include procedural and dynamic aspects, forming a basis for *object modeling* in the business environment

The first section of the book describes the need for and perspective of information models. Mr. Bruce provides a brief history of design methodologies and their limitations and the evolution of IDEF1X from the early work of Robert Brown at Bank of America. He concludes with a discussion of John Zachman's framework and its role in understanding the needs and viewpoints of each of the stakeholders in an information model. The key point made by Mr. Zachman and reinforced by Mr. Bruce is that each model is describing the same thing, namely the business. The different models reflect the individual stakeholder's specific requirements and areas of responsibility but not the differing levels of detail. IDEF1X models are excellent at describing a business's data perspectives, while other methods and techniques are needed to describe the functional (how) and network (where) viewpoints.

The second section discusses the components and development of information models. Each topic is introduced with a clear and concise definition followed by discussion, examples, and exercises. The appendix contains answers to selected questions plus something I have never seen before: an explanation of the answer. This innovation allows you to check your understanding of the text and compare your reasoning with that of an expert modeler. The discussion of entity types, attributes, and relationships is straightforward and holds no surprises. Mr. Bruce does emphasize the role that accurate names and definitions play in developing a model that satisfies the business requirements of our customers. This critical and often overlooked topic is explained by working through several examples demonstrating both good and bad names or definitions.

The book really stands above its peers when generalization hierarchies (super-type/sub-type) are introduced. Starting with common traits among entity types, the author takes you through development and refinement of a generalization hierarchy. At each step he provides insights and tips for ensuring the quality of the model and points out pitfalls and common mistakes. This discussion would apply equally well to the development of a class hierarchy in an object-oriented environment. He completes the discussion of generalization hierarchies by describing the limitations of the current version of IDEF1X and possible extensions to the language. The second section concludes with the best discussion of business rules and normalization I have seen published.

The discussion of normalization begins with Mr. Bruce's mantra: "One fact in one place—get the business rules right."

Normalization is presented as a practical business goal and not as a mathematical concept. The chapter is built around the example `EMPLOYEE <has> zero or mode CHILDs`. Mr. Bruce develops a model demonstrating the features and problems of each normal form (1 through 3) and demonstrates the very real problems that (de)normalization can cause. This deceptively simple example highlights issues such as "overloading" attributes, multiple occurrences of the same fact, conflicting facts, loss of facts, incorrect facts, and confusing facts (entity/attributes) with relationships.

His discussion of business rules proceeds from the negative, finding and correcting hidden errors, which are buried in a model resulting from mechanical application of normalization or "expert advice." Examples of advice include "always use surrogate keys" or "never use identifying relationships." This poses an interesting question regarding the object communities near unanimous use of object IDs: "How are business rules captured and enforced in these systems?"

The third section deals with issues surrounding the use of in-

- Interchangeable parts can be produced by multiple vendors, increasing competition and quality while lowering prices. This is the "open systems" promise.
- Assemblers and maintainers do not need nearly the skill of creators. For example, many people can change their own oil and perform other minor maintenance without a professional background in auto repair and engine tuning. This means organizations can lower the cost of software maintenance and reduce the risk of becoming dependent on small numbers of skilled professionals.

TAKING IT TO THE NEXT LEVEL

Now that we have established a basic understanding of where we are, where we are going, and what objects have to do with it, we can proceed to a deeper examination. What can we learn from the manufacturing model to help us create the appropriate infrastructure? The most important difference between the current software industry and a manufacturing industry is that a manufacturing organization places primary emphasis on formal communication and specification of a product's behavior. This essential step enables the creation of large integrated products from multiple-source components. In producing a car, for example, every part is formally specified in a document used for formal communications, from the diameter of the rivets to the tolerance of the door hanging. Every process used to manufacture the auto, and every piece of knowledge necessary for maintenance, is formally specified and communicated both internally and externally.

It is precisely this attention to formality and detail that enables the creation of interchangeable parts.[2] If I know an engine's temperature range, electrical characteristics, etc., I can produce a suitable set of replacement spark plugs.

How different this is from today's software market! Can you imagine Lotus making public the exact internal specifications of its spreadsheets so I could create an independent replacement for the recalc engine? Or Borland doing the same so I might actually produce an engine that could work on both Lotus' and Borland's spreadsheets? The same goes for Microsoft.

This is not a judgment on Lotus, Borland, or Microsoft but a reflection of current industry mechanics. Current spreadsheet products are large proprietary packages and vendors have no reason to change overnight. But as objects are used to construct the next generation of spreadsheets, change will occur. Objects represent a natural move toward a manufacturing model. They are based on formal specifications (protocols), and formal communications (messages). Next month I will continue this discussion with: how objects in languages and databases relate to the manufacturing model; where standards fit (and don't fit); and how organizations can maximize their ROI from this industry change.

References

1. Stewart, M.K. The natural history of objects, "Happy 25th Anniversary Objects!," special supplement to C++ REPORT, SIGS Publications, 1992.
2. Cox, B. Planning the software industrial revolution, IEEE SOFTWARE, November, 1990.

Tim Andrews is Chief Technical Officer of ONTOS, Inc., and may be reached at ONTOS, Inc., Three Burlington Woods, Burlington, MA 01803, by voice mail at 617.272.7110 x288, or via email at andrews@ontos.com.

OBJECT METHODS ≡≡ *continued from page 12*

nology transitions. From a personal perspective, many software professionals are used to working more independently than the object-oriented paradigm encourages. Things like CRC designs or code walk thrus require a different perspective. We have to create new roles and change old ones. Some old practices must be discarded and new ones implemented. We must accept that less is better as well as readjust our reward systems for output that is produced by one individual but used for others. The "experts" of the traditional world are most likely not experts in the new world and may never be.

SOME STEPS TO MAKE IT EASIER

I believe there are several steps to make the transition easier. In general, recognize that the tasks people perform every day will change. Understand that a combination of training existing personnel and access to experienced personnel is key to success. Learn to value cooperation and sharing among employees, expect that reward structures need to be reviewed, and anticipate that new roles will exist. Some other specific actions include:

- Commit to walk thrus.
- Create and support a quality philosophy.
- Create naming conventions.
- Recognize the role of good class documentation.
- Commit to stable interfaces.
- Provide access to code, documentation, testing cases, tools, and plans.

CONCLUSION

The transition to object-oriented paradigm is, and has, paid off handsomely. Successful transition opens up tremendous opportunites for organizational optimization. Fewer people do more or the same number of people accomplish a lot more. It also measurably improves the "time to production." It is not an overnight process. Success is usually accomplished in years, not months, but it is worth the effort.

Patti Dock is Vice President at Orgware Inc., consulting with organizations as they migrate to object-oriented techniques. She teaches a course called OBJECTMethods, which compares and contrasts leading object-oriented design methods. She can be reached at 203.270.1242.

O-O transitions require cultural changes

Patti Dock

Over the past few months, I have worked with several companies as they move to an object-oriented paradigm. One of the most interesting aspects of this transition is the cultural transformation that must occur. Whether it's a high-tech engineering firm or a large corporation with a traditional MIS organization, most companies find it difficult to grasp cultural changes. From an organizational aspect, this change is not easy or fast. Success requires commitment from the highest levels in the organization down as well as acceptance from respected technical proponents.

WHY THE CULTURAL CHANGE IS NECESSARY

Object-oriented transitions are not about languages or hardware or methodology selections. Although each of these selections plays a key role, successful transitions are centered around restructuring what individuals do on a daily basis. Everything—from an organization's structure to its programming languages, idioms, style, tools, methods, development practices, and even its project management approach—will directly affect the benefits derived from this shift.

The extent to which a corporation takes advantage of reuse is key to maximizing the potential benefits of an object-oriented transition. Reuse implies sharing and trust: both qualities we have espoused in software engineering for decades but often have not practiced. How can people reuse code without being reasonably certain it works as documented? How can they use code if it is not documented at all? How many times will they reuse code if the first time they try it doesn't work correctly or, more subtly, if it doesn't work the way they expected it to?

WHAT CULTURAL CHANGE INCLUDES

Cultural change encompasses code walk thrus; use of outside talent; and visible organizational commitment to quality, testing, naming conventions, documentation, and stable interfaces.

The cultural changes required for successful walk thrus are threefold: 1) simple understanding and acceptance of public review of design and code, 2) acceptance that less is better, and 3) disciplined adherence to naming consistency.

In many organizations, people simply do not perform code inspections or walk thrus. The idea that people outside your project team will examine your code and publicly comment on it is new. This is a difficult obstacle to overcome. I recommend that organizations commit energy and resources to developing, documenting, training, and implementing a standard approach for walk thrus so that the experience is consistent, constructive, and useful to the developer. In other groups, the most difficult hurdle is the idea that "less is better." Many organizations still reward people for amount produced instead of amount reused. An interesting observation from sitting in on many design and code walk thrus of object-oriented systems is that, especially in a mature environment, the amount of software that comes in shrinks before it goes out. That is, the group's participants most often have suggestions for reusing existing software or making it smaller.

The practice of rewarding based on amount produced rather than amount reused must change for successful implementation. Reward must not go just to individuals who reused the class but also to those who designed and implemented the reusable class.

Classes and methods are often named from too narrow a perspective. This, as well as areas where naming can come in line with already existing classes, is often discovered in walk thrus. The amount of change likely to occur in the naming of classes and methods is therefore high. Time spent on name selection is time put to good use; the probability of a class being reused is directly tied to the proper naming of the class and its methods.

EXPECT TO NEED OUTSIDE HELP

Object-oriented expertise grows with experience. It is not unusual to require two to three projects before someone really produces good object-oriented design and implementation. The more access developers have to experienced individuals, the quicker the transition. Whether your organization hires experienced individuals or hires external consultants, access to experienced individuals will improve your position on the learning curve.

Some organizations traditionally use external consultants and some regularly hire from the outside. In cases where either of these actions is the exception rather than the rule, the organization must change its practices. The cost of growing internal expertise in a vacuum exceeds the cost of bringing in skilled people—and the resulting quality is better.

Another type of "outside help" is interproject movement. The more experienced members from one project are used as internal consultants to seed new projects.

QUALITY PHILOSOPHY

Introduction of any technology creates opportunities for improving the development process. If the organization is invest-

ing in a transition to object-oriented software, take advantage of this opportunity and implement a high-quality object approach. With that in mind, it is worth noting some of the object-oriented quality lessons learned in commercial projects. If we hope to evaluate quality improvements, we must measure quality. Encapsulation eliminates side effects but inheritance introduces new testing requirements. Testing will require as much effort as development—a lot more if proper tools are not available.

Development of high-quality object software requires a culture in which both objective and subjective quality are routinely measured at every stage of the development. This means you must measure the extent to which you meet the specification as well as the extent to which you meet the needs of the user. Often these are two different criteria. If you are expecting to produce reusable components (which we highly recommend), you must also readjust who the user is—sometimes it is another programmer.

Objects reduce the probability of many kinds of errors. This is attributed to models that are more accurate and easier to understand as well as to encapsulation, which isolates how we do something from what we do. However, a less discussed phenomenon is that inheritance actually breaks the encapsulation and therefore introduces unique quality requirements.

Investing in testing tools is a new concept for many organizations. (The most common exception seems to be successful networking projects that traditionally invest heavily in testing tools.) Because of inheritance, projects need to verify that all inherited methods used by a class are correct, that subclasses of the specified argument type are correct, that all methods by the same name perform the same logical operation, and that the documentation is accurate and sufficient enough for an isolated user to use the component. This can be a daunting set of tasks if the organization does not invest in testing tools or, for that matter, test cases that live with the class.

PROJECT SCHEDULE

Most proponents of object-oriented technology espouse the need for naming conventions, documentation, stable interfaces, and an iterative approach, but few organizations have either the culture or the structure to support these changes. The structure must accommodate these changes and the culture must adapt to allot appropriate time in the project schedules.

Naming Conventions

Naming conventions have always been important in software engineering, but polymorphism and widespread reuse increase the need for well-thought-out, consistent naming. Time spent thinking about names for classes and methods is time well spent. Organizations must learn to budget time for this in their project schedule and enforce adherence to standards.

Documentation

Documentation for classes is very different in an object-oriented system than in a non-object-oriented system. The most important measure of a well-documented class is that it's descriptive enough for users to recognize that it fits the system requirement, and accurate and detailed enough for them to utilize the component.

Stable Interfaces

Stable interfaces don't come automatically. One expects the interface of the first iteration of a class to change. Before a class is "certified" as a reusable class, its interface must have reached a maturity level that ensures its relative stability. Upon being certified reusable, a new set of rules that are well thought out and viable must arise. Modifications are premeditated, often publicly debated, certainly well thought out, and most definitely well-publicized actions. When deletions must occur, be sure to understand the impact. If you treat the stable interface lightly, project teams will burn out and be wary of reusing classes again.

Iterative Approach

Most organizations do not use an iterative development approach although some have experimented with rapid development. One of the largest differences in almost all successful object-oriented transitions is the use of iteration. The difficulty lies in convincing management that the software is not ready to ship because it is up and working but requires another iteration. Again, companies must learn to budget time for iterations in their project schedule.

> ***Development of high-quality object software requires a culture in which both objective and subjective quality are routinely measured at every stage of the development.***

ACCESS TO CODE, DOCUMENTATION, TEST CASES, TOOLS, AND PLANS

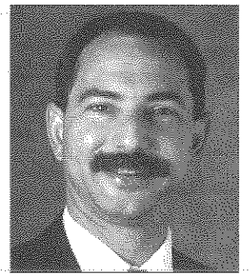
Real structural changes are required if a corporation wants to encourage reuse. It is no small task to solve the configuration control issues associated with the code, documentation, test cases, and test plans for all the projects. If these are not accurate, up-to-date, and readily available to individuals in a timely manner, they won't be used, which is counterproductive. The importance placed on these activities and the respect accorded those playing key roles will be proportional to the success of cultural changes throughout the rest of the organization.

WHY IS IT SO HARD?

Change is difficult. People naturally resist change. But the object-oriented change is even more difficult than many other tech-

continued on page 14

Object databases, software ROI, and the movement toward a manufacturing model for software



Tim Andrews

A CHANGE OF EPIC PROPORTIONS

There is a great movement under way that is changing the way software is created, distributed, and maintained. The software industry is attempting to move toward a manufacturing model, where "cookie cutter" software components are created and distributed in large numbers and then assembled into ever larger systems. This represents a radical departure from the current model, which is much closer to the print media industry: Software is "written" in self-contained "packages"; the "author" is often paid a "royalty"; and the software "publisher" hopes for a "best-seller."

I firmly believe that this movement toward a manufacturing model will have a profound and positive effect on the entire software industry and its customers. Further, I believe that object technology, and object databases in particular, will play a crucial role in this industry reorganization.

IF YOU DON'T KNOW WHERE YOU ARE, YOU CAN'T KNOW WHERE YOU'RE GOING

Software creation today is still largely a craft, with large pieces of software being written by small numbers of highly skilled individuals. A much larger body of professionals creates the final package, testing for quality, documentation, marketing, sales, distribution, and customer support and logistics of the software product.

This model has significant advantages:

- The cost of entry is fairly low. Almost anyone or any small group can produce a software package. The emphasis is on creativity and innovation, and new product development occurs at a furious pace.
- The potential return is enormous. With very low overhead for production, a successful software package generates an obscene profit margin. Despite the emergence of huge companies such as Microsoft, every year small companies experience phenomenal growth when they introduce a successful product into the market. (A good example is Powersoft, a company virtually unknown two years ago that posted approximately 20 million dollars in 1992 revenue.)

THE WORLD OF SOFTWARE MANUFACTURING

Now let's examine the "software manufacturing marketplace." A true, high-level manufacturing market is almost the opposite of the current software market:

- The cost of entry is very high. Large initial investments are required and a substantial organization must be in place before the product can ship.
- The return is average but predictable. Because of a much larger initial investment in infrastructure, product must be amortized over a much longer time period. In other words, you have to sell a lot more before you break even.

In a large manufacturing environment, such as the auto industry, there is a fairly even set of skills distributed through all phases of the product lifecycle. Manufacturing planners are perceived as peers to the designers and planners—there is no hero worship. Who is the Phillipe Kahn of Mercedes Benz? The Bill Gates of Honda? They do not exist because the industry is not organized around the creative talents of a small group. That is why such industries are more predictable.

SO WHAT DOES ALL THIS HAVE TO DO WITH OBJECTS?

Objects, by their very nature, encourage the manufacturing model. They have the characteristics of components, like IC chips or parts, and they are more like manufacturable products than programs. One of the primary notions of object technology is the "virtual machine." Alan Kay recently described this very eloquently:

> **The shock of realization was so great that it was the last time I ever thought in terms of subroutines and data structures. I could see right away that you didn't need to divide computers into anything less simple than other computers, and that...virtual machines...were exactly the right way to go.[1]**

It's easy to see how one can envision a market of off-the-shelf software components made out of objects. Each object component is a self-contained machine that performs specific functions within predefined constraints.

The benefits of this model to the customer are great:

- Standard parts will be produced at lower costs, lowering the cost of software.