NEW DIRECTIONS IN SOFTWARE 1960-1966

BY

ASCHER OPLER

Reprinted from the PROCEEDINGS OF THE IEEE
VOL. 54, NO. 12, DECEMBER, 1966
pp. 1757-1763

New Directions in Software 1960–1966

ASCHER OPLER

Abstract—**This article reviews selected significant developments in the rise of software to an equal partner to computer hardware. Topics covered include transition problems, hardware–software interdependency, control programs, programming languages and their processors and certain data-file management aspects. The period October 1960 to April 1966 is covered.**

AT THE TIME when the first special computer issue of these PROCEEDINGS appeared (October 1953), the subjects to be reviewed in this article were given scant treatment. Only the paper of Hopper and Mauchly [1] appears relevant. When the second special issue appeared (January 1961), an article by Orchard–Hays [2] covered the "state-of-the-art" as of October 1960. Since that date, there has been a technological revolution resulting in the transformation of the position of software[1] from that of *a useful aid* to programming and operating computers to that of *an equal partner* to hardware in terms of importance in computer technology.

This article will *not* attempt to review and evaluate all developments in this field between October 1960 and April 1966. Instead, the current position of software will be summarized and four significant areas will be covered in some detail. This delimitation is necessary since the scope of work in this field is far beyond inclusion in a summary article. Indeed, since 1960, there have been at least ten major special conferences on aspects of software, the publication of more than one hundred papers and the appearance of at least a dozen books and monographs on software topics. In a report date 1963[3], 52.1 percent of a sample of 549 members of the Association for Computing Machinery listed software development as their primary activity. In 1966, in the United States alone, an estimated 5000 programmers are involved in the production of software.

The article will concentrate primarily on developments in the United States although activity reported in other countries represents a significant contribution. The International Federation for Information Processing has provided means for effective international communication via two major conferences (IFIP Congress—1962 in Munich and IFIP Congress—1965 in New York), a series of special conferences on programming languages and their processors and the work of IFIP committees.

Only general-purpose software for general-purpose computers will be covered. The development of computing systems for *dedicated* purposes (e.g., telecommunication, process control, reservations systems) has stimulated the development of software with the requisite attributes of reliability, response time, and channel capacity. While control programs for such applications have many features in common with control programs for computers used to handle a variety of computational tasks, discussion of their special problems cannot be included here. However, several other papers in this issue consider relevant aspects of such software. Furthermore, time-sharing software is covered elsewhere in this issue.

Manuscript received August 29, 1966; revised October 10, 1966.

The author is with Computer Usage Education, Inc., New York, N. Y.

[1] For this article, software is defined as the complete set of control programs, language processors and utility programs generally supplied to complement computing equipment (hardware). Specific application programs are not considered software for purposes of this article.

SOFTWARE PRODUCTION

The five and one-half year period covered has seen the development of a production process for software fabrication. In a talk presented early in this period, the author [4] pointed out that a software crisis was at hand because:

1) Quantity requirements were rising.
2) Product delivery was lagging.
3) Product quality was unsatisfactory.
4) Product costs were exceeding estimates.
5) The shortage of qualified designers and implementers was growing.
6) Proposed methods for automatic software production were not fulfilling expectations.

Today, software production remains a problem area. The major computer manufacturers have all found it necessary to abandon the practice of assigning the responsibility for producing software systems to conscientious programmers with the requisite technical competence. In re-evaluating the production problem, the management of each computer company has generally come to the same set of conclusions.

1) Production of hardware and software should be carried out under single unified responsibility. Software should no longer be regarded as "sales support" produced by the marketing division.

2) Production of software should be carried out at the same site as production of hardware.

3) Plant production technology should be applied to software. This calls for development of prototypes, specifications, quality control, schedule control, cost control, acceptance testing, etc.

4) Although this technology forms an effective production *framework*, qualified implementers constitute the *core*. To obtain the services of programmers with sufficient technical competence, large-scale recruitment and training should be undertaken.

5) Automatic software production technology should be developed and when perfected, put into operation.

This transition to factory-type production methods is still in progress. Completion is a few years away, but those manufacturers who have installed such methods appear satisfied with all results except costs, which have mounted beyond all projection exceeding, in many ways, hardware production costs. To date, automatic production of software has not been successful in turning out a product of commercially acceptable quality. This lack has been partially compensated for by the application of computers to many other processes involved in software production scheduling, specification, check-out, testing, documentation, etc.

Transition to Next-Generation Computers

The development, marketing, and installation of second-generation computers (transistors and printed circuits) was completed during this period and the introduction of third-generation (microelectronic circuitry) computers initiated. That problems exist in adapting applications programs for a next-generation computer has always been known, but it is only in the past two years that most people have become aware of a) the size of the investment of man-years in the existing pool of programs and, b) the difficulty of executing a smooth, minimum effort transition.

There are a number of known aids to transition between computer generations. For second-to-third generation conversion, the following are available:

1) Simulation of the old computer on the new computer.
2) All-hardware compatibility mode in which special circuitry permits the new computer to perform all functions of the old.
3) Emulation in which certain functions of simulating the old computer are performed by special circuitry while others are performed by software.
4) Recompilation of programs written in those programming languages for which processors are available on both the old and new computers. Where the two versions of the programming language have significant differences, language version to language version translators may be useful.
5) Data conversion aids which assist in reformatting data, changing character sets, labels, etc., as required by the new computer and its software.

The one method that is absent is that of fully automatic translation of any program (existing in either machine code or assembly language) to programs that can execute successfully on the new computer. While considerable effort has been spent in the development of such programs (8, 9, 10), the only translations in successful use today are special cases:

1) Translators operating between two computers with relatively small differences in logical design (e.g., Honeywell 200/IBM 1401).
2) Programs which translate the simple equivalences of a program and flag the difficult portions (11). These are sometimes called "reprogramming aids."

At the present time, the two most promising conversion routes are 1) via emulation and/or all-hardware compatibility, and 2) via use of standard programming languages. However, past difficulties are expediting the current movement toward the *mandatory* use of such languages. Concurrently, language standardization efforts, long in progress, are resulting in standardization at a most propitious time.

Hardware-Software Interdependency

For many years, despite claims to the contrary, there has been little significant interchange between logical designers and designers of software systems. To software system designers, it appeared that hardware designs were facts-of-life to be surmounted by their product. Computer designers blamed software designers for masking the beauty of the logical design with programming systems.

The direction that computer design has taken has forced an end to this isolation. The design of the Burroughs B-5000 (1961) was clearly the product of joint efforts. The computer aided the language processors by providing push-down storage, while the logical design necessitated a specialized control program in order to operate. This pioneering effort pointed out:

1) That software performance could be enhanced by inclusion of appropriate manipulative ability in the order set.
2) That software designers could build processors aimed at taking advantage of such facilities.
3) That an unconventional logical design made a special control program an *integral* part of the whole hardware-software system.
4) That the logical design forced the adaptation of a rigorous set of programming techniques and conventions.

Since that time, increasing interchange—even teamwork—has often occurred. Consolidation of the two developmental efforts under single management has facilitated this interchange. The knowledge that the two groups could really support and encourage radical departures led to a new and better mutual appreciation. Based on the logical equivalence of hardware and software, careful and economic trade-offs have begun to appear. For example, if the cost of main core storage can be drastically reduced, large complex control systems may reside in memory replacing expensive complex circuitry, provided that the control functions can have fast enough response. This illustrates the type of interaction that requires considerable joint analysis to reach proper decisions.

Third-generation computers generally have less control circuitry than earlier computers because of greater reliance on control programs. The recognition of the importance of control programs is borne out in the design of a two-mode computer which prevents "ordinary" programmers from altering control programs stored in memory. In the event