Guest Editor's Introduction

William F. Ross
M & S Computing, Inc.

The dramatic decline in computer hardware costs in recent years has brought the costs of software development into sharp focus. But important efforts to address software technology have been underway during the same period—notably in the 60's by Bohm, Jacopini, Dijkstra, Parnas, and others in the university environment, and in the 70's by such people as Mills and Baker, who defined specific elements of a methodology directed toward reducing software costs and improving software quality. This methodology, which according to Mills and Baker was successfully applied to the New York Times information bank project, is generally referred to as structured programming (SP)—but includes such methods as top down design and chief programmer teams.

Subsequently—and inevitably—all this attention prompted questions as to whether SP is in fact feasible and economically practical on an industrywide basis, and if so to what extent. This made it a timely topic for the 1974 Lake Arrowhead Workshop.

Sponsored by the Western Area Committee of the IEEE Computer Society, this annual workshop focuses each year on an emerging area of engineering activity. The 1974 workshop was especially significant from at least two standpoints: it was the first time a programming topic had been selected as a Lake Arrowhead Workshop theme, and it constituted an implicit recognition of structured programming as a software engineering activity. The latter is an important step towards achieving the discipline needed to make software costs and quality behave in a manner similar to that of computer hardware.

Workshop Organization

To determine the industrywide applicability of SP, the workshop was slanted toward applications and objective descriptions of quantitative results observed, with representative participation by leading practitioners from all elements of the programming community. The breakdown of workshop participation was 20% university, 15% government, and 65% industry.

The agenda was partitioned into four sessions: (I) Concepts and Definitions of SP as a Scientific Discipline, chaired by John Naughton of IBM; (II) Experience and Accomplishments Resulting from Applications of SP with Emphasis on Measured Results, chaired by Dr. Barry Boehm of TRW; (III) Problems, Approaches, and Techniques Associated with the Advancement, Development, and Implementation of SP, chaired by Dr. Robert Brown of Hughes Aircraft Company; and (IV) Impact of Structured Programming on Evolving Technologies and Related Programming Disciplines, chaired by Dr. Guy de Balbine of Caine, Farber and Gordon.

The Lake Arrowhead Workshop, like most other workshops of its type, does not publish a proceedings, nor does it permit sessions to be taped or viewgraphs photographed. The purpose of this restriction is to encourage a maximum of informal, candid, lively interchange on current topics of technical interest without the fear of violating proprietary rights or the burden of formal documentation of experiment results.

However, to make some of the high points of this workshop available to a wider audience, each session chairman was asked to summarize his session and (with the authors' permission) to present selected papers that were representative of his session. A similar procedure had been followed the year before (see the March 1974 issue of COMPUTER).

Because each chairman was given a free hand in summarizing his own session, there are inevitably differences in treatment. Some chairmen chose to present fully-developed papers from one or two of the speakers; others preferred to present short statements from all of the speakers, integrated into an overall session description. The net effect, if not one of homogeneity, will at any rate be genuine.

Some Observations

During the organization of the workshop, a number of questions arose as to what structured programming means. At first I was tempted to restrain the scope of discussion by selecting a narrow definition such as structured code or top down design. Finally, it was left to the workshop participants themselves to provide insight into what SP means—at least to the segment of the programming industry that professes to practice it. The participants readily adopted a broad definition, including not only structured code, top down design, chief programmer teams, HIPO, modularity, structured design (sometimes called composite design), but also derivatives and related programming functions and methods that could be reasonably associated with the foregoing.

During the course of the sessions, many experiences were presented that verified the positive results already claimed by IBM—as the papers will attest. However, significant problems were also identified in the application of SP to achieve all of its goals. For example, Chuck Holmes of McDonnell-Douglas Automation Co. and James Romanos of GTE Sylvania provided some excellent insight regarding the practical problems involved in getting SP started in programming organizations.

The general consensus was that those practicing SP are still in the minority. However, the experience base is widespread. Diverse elements of government, universities, and industry are all successfully applying the methodologies to their work. This is illustrated by the representative sample of participants who contributed to this issue of COMPUTER. The better defined methods—e.g., structured code, chief programmer teams, etc.—have been applied to scientific, business, control, and real-time programming systems with equal success. The workshop participants were enthusiastic and even religious regarding the practice of SP. Of course, if a religion exists there must be a heretic somewhere. Fortunately, there were a few at the workshop and their views received much attention. In Session IV, for example, Dr. Noah Prywes of the University of
Pennsylvania suggested that SP will be outdated before it is put into common practice.

No one stood up to defend flowcharting of programs as a means of documentation. There were a number of contending approaches, e.g., HIPO, PDL, etc., but none present advocated program flowcharting, or even alluded to it.

There was a great deal of discussion of program design languages (PDL). A little time was spent determining exactly how many extensions of the fundamental language forms, i.e., if-then-else and do-while, may be practical. However, the bulk of the time on the subject of PDL was devoted to a number of approaches to the specification of programs via new language approaches. The material in this issue describes some of the approaches discussed.

For technical problems, the uppermost problem on the minds of the workshop participants was "structured data." There were no ready solutions to the formalization or standardization of data definition such that the same benefits could be achieved as structured code provides. In fact, the problem was identified by many participants along with a plea to establish some practical approaches and guidelines for immediate implementations along with structured code.

On the subject of structured design there were a large number of advocates. L. Constantine's name occurred very frequently, and a large number of the workshop participants were not only familiar with the design approach, but were enthusiastically practicing it. Preliminary results were highly promising, but evaluations were not as complete yet as for the other techniques.

Some Opinions

A recurring question at the workshop was, Which of the SP techniques provided the greater benefits? Although many speakers presented extensive experience and attested to significant benefits in employing SP, none was able to quantify each specific technique's incremental contribution toward the totality of benefit. Since this question remained unanswered, I feel compelled to venture an opinion that may at least stimulate controversy:

Top-down implementation—i.e., code a little, test a little—seems to produce the majority of the increase in programmer productivity. This opinion is primarily based on experience with three recent large scale real-time system development efforts at Hughes. The first real-time system development project exclusively used a top-down program implementation approach and a program production librarian to control the baseline system tape. High programmer productivity was achieved. This represented approximately 50% dollar savings over the productivity of the next best project, based on over eight years of historical data. Similar programmer productivity was achieved for a second project that used top-down implementation as well as chief programmer teams, HIPO, librarian, and structured walkthroughs. However, a third project implemented during an overlapping time period used a "pure" bottom up approach, i.e., the total program was coded completely before any debug or testing was accomplished on the computer. This project employed structured code and a librarian, but it had very poor programmer productivity in comparison.

Of course, the environment for these three projects was not controlled. It probably never will be controlled for projects of one to two years' duration with more than 6 programmers working in an R&D engineering environment. Nonetheless, I have been very impressed with top-down implementation as a means to provide timely feedback to programmers that their efforts to date properly fit into the overall system through demonstration. This assurance along with the precise definition of the interface for the next program increment to be implemented appears to be the factor that eliminates long integration phases, and thus results in high programmer productivity.

If top-down implementation is in fact responsible for most of the improvement in programmer productivity, then why use such methods as HIPO's and structured walkthroughs? If we review Barry Boehm's concluding remarks from Session II, we find that high programmer productivity in itself does not solve all programming problems. The additional use of HIPO's, walkthroughs, etc., by the second project described above appeared to lower design costs somewhat, but more important, their use facilitated communications during the development phase between programmers and other disciplines such as managers, system engineers, hardware engineers, and users.

Principally through use of HIPO's and HIPO walkthroughs, the programs were well understood by all disciplines during the development phase and operated exactly as expected during competitive field trials. As a result of field trials that lasted approximately four weeks, only six changes were requested by the ultimate user. All six changes were incorporated and retested in two working days. This short turnaround period is indicative of how closely the programs performed to expectations.

Summary

Experience reflected by workshop participants indicated that all elements of SP work for all programming environments. However, the practice is often limited to select groups within a larger organization. Adoption of SP techniques is not without problems; however, it is recognized as the initial step toward lower programming costs and higher quality. SP is expected to spawn a multitude of new disciplines, perhaps even before SP as we know it matures. The key to success of SP rests with the positive results achieved to date coupled with the commitment to rapid advances in software technology.

William F. Ross chaired the 1974 Lake Arrowhead Workshop on Structured Programming. He is Manager of Advanced Programs with M&S Computing, Inc., Huntsville, Alabama, where he is responsible for development of advanced software and interactive graphics concepts for space program applications. Prior to joining M&S Computing, he was Manager of Advanced Software Technology at Hughes Aircraft Company, Fullerton, with responsibilities for software R&D and application of structured programming methods to development of large scale real-time systems. Ross entered programming in 1958 with RCA and has served in both technical and managerial capacities with GE, NAA, and IBM. He has worked in all phases of program development and has lectured at UCLA on simulation modeling and structured programming. Ross received his BSEE from Drexel University in 1958. He is active in the IEEE Computer Society and ACM and is a member of Eta Kappa Nu.