billion in 1985. There is a need to reduce the cost of software development, as compared to 1970 technology, by 80%, in order just to keep in step with the increases in revenues from hardware. Cost reductions from the SP discipline have been widely estimated during the workshop, in the range of 0-33%. The conclusion is not that SP is inadequate, but that the impetus for new software technology will continue and will engulf SP.

How will new technology alter structured programming? Consider first the example of new operating systems technology, illustrated by MULTICS. Due to the facilities for independence, sharing, and interprocess communications, much more responsibility and freedom could be accorded to the team programmers. Namely, the chief programmer will have less to do preparatory to assigning work to the members of his team. He will not be concerned with memory layouts and program linking and loading. Because of the dynamic resource allocation he will not be concerned with hardware considerations. In his system he can freely incorporate parallel processes which can be executed in a large scale multiprocessing environment, which will be very helpful for real-time systems assignments. On the other hand, he will have new responsibilities at the end of the programming activity. Efficiency of such systems can be greatly improved through tuning, which consists of binding or partitioning components of the system—be they data segments, procedure segments, or entire processes.

Expected progress in automatic programming can be summarized as follows: Roughly speaking, the 1940's brought the automation of execution of instructions; the 1950's brought supervisor macros and assembly languages; and the 1960's brought high-level and problem-oriented programming languages. In the 1970's we are looking toward automation of system and module specification languages. Figure 3 illustrates this idea, showing at the center a business program module generator which accepts functional specification as input, and produces a high-level language program. As shown, the generator itself can be generated and modified automatically by the use of special generators that produce the components of the former. It appears that such systems would be problem oriented, and different problem areas would require modified generators. However, each generator could be applied to a wide area of applications, such as for business programs.

Systems of this type are being developed at the University of Pennsylvania for business, econometric simulation and forecasting, and automatic testing of electronic and automotive devices and systems. The specifications prepared by the chief programmer could be fed directly to the automatic system to produce the programs. Therefore, there would be no need for team members to compose programs. The chief programmer would be able to produce an entire large-scale software system. However, he would need assistance in integrating, testing, installation, and updating of the software. The role of the programmers will also change, therefore, as compared to current practices of structured programming.

Conclusions

Guy de Balbine
Caine, Farber & Gordon, Inc.

The main trends discernible both in Session IV and in the general discussion that ended the 1974 Lake Arrowhead Workshop can be summarized as follows:

Languages Implementation languages will evolve through improved control structures for structured coding, language facilities to help prove the correctness of programs, extension of existing compilers via preprocessors, and standardization of new features.

Design languages will emerge via designing in structured pseudo code, computer-assisted tools for design, and methods and/or tools to structure data.

Finally, design and implementation languages will be integrated.
Systems to Support Structured Programming Program development and maintenance aids will be developed, including tools to perform and audit top-down refinement, and project-oriented data bases—an extension of the current program production libraries.

Improved operating systems will be developed, and hardware support will be achieved through an appropriate choice of the underlying machine architecture.

Structured Programming Methodologies SP methodology improvements will include formal design methods, computer-aided design and verification systems, and evolution towards automatic programming.

Transition to Structured Programming In the field, the spread of SP will be aided through consolidation of current efficiency and reliability gains, management commitment to update both the methods and tools, and development of transitional tools to help the conversion.

Educational advances will include teaching how to solve problems, and how to design rather than simply how to code.

Further Ahead Other improvements will include requirements and specification languages, and automatic program generators.

Guy de Balbine is Technical Director of Caine, Farber & Gordon, Inc., a Pasadena firm specializing in the construction of large software systems. He has been responsible for the conception and development of the “structuring engine.” Previously, he was a senior scientist with AERO, a French R&D organization.

Dr. de Balbine is a graduate of Ecole Centrale (1963) and received his MS and PhD from Cal Tech in 1964 and 1968, respectively.

Charles Zahn is on the staff of the Computation Research Group at the Stanford Linear Accelerator Center. From 1972 to 1974, he was a visiting scientist at the European Center of Nuclear Research in Geneva, Switzerland. His research interests include two-dimensional pattern recognition, unsupervised cluster analysis, graph theoretic algorithms, and design of programming languages to support structured programming. He received the BA and MA degrees in mathematics from the Catholic University of America and the University of Wisconsin at Madison, respectively.

Lawrence Robinson, a research engineer in the Computer Science Group at Stanford Research Institute, specializes in software methodology, formal specification, operating systems design, and program verification. He has taken the leading role in the development of a methodology for the design, implementation, and proof of software systems that is being applied to several large systems at SRI. He received his BS from Yale in 1971, and is currently completing his PhD dissertation, “Proof Techniques for Hierarchically Structured Programs,” at Carnegie-Mellon University.

Richard Bigelow has been employed at Burroughs Corporation in Goleta, California, since September 1973. His principal interests are programming and applications languages. Previously, he was a research fellow at the California Institute of Technology, where he received a BS and PhD in engineering science.

References


Judy A. Townley has been on the faculty of Harvard University for the last two years, teaching both undergraduate and graduate courses in the areas of software engineering, aspects of programming and programming languages, and theory and development of compilers. She is currently involved in the semi-automatic selection of representations for data and the development of a new programming system for the construction of large, complex programs. She is also director of a new master’s program in information science at Harvard. Dr. Townley received her BA from the University of Texas at Austin in 1967 and her PhD from Harvard in 1973.

David J. Farber is an associate professor of information and computer science and electrical engineering at the University of California, Irvine. He is also a principal investigator of the NSF-supported Distributed Computer System Project (DCS) at UC Irvine. Earlier he was a supervisor of computer research at Bell Telephone Laboratories, where he was active in electronic switching and computer communications studies and was a co-author of the SNOBOL language. Farber’s professional activities include membership in the IEEE, the IEEE Computer Society, the IEEE Communications Society, the ACM, and Sigma Xi. At present he is vice chairman of the ACM’s Special Interest Group on Computer Communications and chairman of the IEEE Computer Network Group.

Alfred S. Liddle is a senior staff engineer, TRW Systems Group. For the past eight years, he has been involved in the development of medium-scale applications programs at TRW’s Sunnyvale Laboratory. During this time, he has been involved in all aspects of program development, particularly requirements analysis, design, testing, and management. Prior to joining TRW, he served for four years as an officer at the Air Force Satellite Test Center at Sunnyvale, California. He holds a B.S. in physics from Brigham Young University.

Noah S. Prywes is an associate professor at the Moore School of Electrical Engineering, University of Pennsylvania. He has published extensively in the fields of solid state computer fault diagnosis, computer machine organization, and information storage and retrieval. He has developed the Multilist file organization which he has used in several large scale applications. Recent work of Dr. Prywes consists of the Problem Solving Facility at the University of Pennsylvania which is an on-line system incorporating interactive programming and information storage and retrieval used for research by students and faculty. He received a BSE degree from the Technion, Israel, in 1949, an MSEE from Carnegie Institute of Technology in 1951, and a PhD from Harvard University in 1954.