Guest Editor's Introduction:

FIRMWARE

The Lessons of Software Engineering

Bruce D. Shriver
University of Southwestern Louisiana

The term "software engineering" was first introduced nearly 10 years ago, at a NATO-sponsored conference held in October, 1968, in Garmish, Germany.[1] Since that time, a variety of tools to assist in the specification, production, and management of reliable software have been proposed.[2] Although we are yet far from the visionary's goal of automatic generation of maintainable code, we are making progress.

At least one of the reasons for this progress is clear: Vast amounts of resources have been spent by government as well as industry, both here and abroad, to develop software engineering tools and techniques. The record of many of these efforts, contained in thousands of pages of technical journals and conference proceedings, reflects a long-standing tradition of shared experiences in the problems of software development: From the earliest days of computing, professionals have regularly and freely exchanged their views and insights; even manufacturers and their customers have joined forces in cooperative efforts to debug large, complex software systems.

Firmware, however, has not enjoyed so happy a tradition. Manufacturers are migrating more and more of the user-visible primitives of their systems down into microcode. As a result, both the firmware and the support and development tools which are a part of the microcode production and maintenance process, now typically regarded as proprietary, rarely appear in the public literature.

And yet, in a great many respects microprogramming is not different from classical programming and could profit from the software engineering techniques developed over the past decade. This issue of *Computer* explores that thesis.

The first paper, by Davidson and Shriver, gives an overview of current firmware engineering practices during microcode design, specification, construction, verification, testing, debugging, and maintenance. It also poses several open questions related to the code production process. The second paper, by Stockenberg and van Dam, develops a method for analyzing the migration of primitives in multilevel interpretive computing systems. A very interesting aspect of this work is the performance prediction aspects of the analysis.

Many early workers in microprogramming proposed that the use of firmware would narrow the then-existing hardware/software gap—ie., reduce the communications problems existing between the machine designer and the programmer. Later workers often conceded that what actually happened was the introduction of two gaps: a firm-

0018-9162/78/0500-0019$00.75

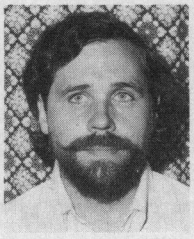
ware/software gap and a hardware/firmware gap. The engineers, microprogrammers, and programmers all spoke different machine languages and used different development tools.

The last two papers are related to the hardware/firmware gap: Barbacci and Parker present a case history of using emulation (and, in particular, microdiagnostics) to assist in the evaluation of architecture descriptions. The paper by Fiala describes some firmware engineering tools that apply to hardware interfaces to a target machine. These papers demonstrate a different set of needs for firmware engineering from those shown in the first two papers. Such a divergence in the contents of these four papers demonstrates that firmware engineering must be sensitive to the issues of top-down design (e.g., language- and system-directed architecture design) while also being responsive to bottom-up design (the use of microprogramming as a processor implementation technology).

Taken as a whole, these papers should introduce the reader to both the soft and hard aspects of firmware engineering. I gratefully acknowledge the help of the authors, referees, and Technical Editor Jack Grimes for their assistance in putting this issue together. ■

References

1. Peter Naur and Brian Randell, Editors, *Software Engineering*, NATO Science Committee Report, January 1968.
2. Ware Myers, "The Need for Software Engineering," *Computer*, Vol. 11, No. 2, February 1978, pp. 12-25.



Bruce D. Shriver is a professor of computer science at the University of Southwestern Louisiana. Earlier he taught and conducted research at the University of Aarhus in Denmark, where he also co-directed a design team in the design of a dynamically microprogrammable processor, MATHILDA. He has also taught at California State Polytechnic University and was a National Science Foundation Fellow for two years. He has had industrial experience with special-purpose computer systems, process control systems, and user-machine interface problems.

Shriver received his PhD in computer science from the State University of New York at Buffalo. He is a member of ACM, the IEEE Computer Society, and SIAM.