

From the Editor-in-Chief

IEEE Software grows from a quarterly to a bimonthly periodical beginning with this issue. The Editorial Board, the department editors and their guest contributors, the authors, the numerous referees involved in reviewing the articles that were submitted to us, and the West Coast Office of the IEEE Computer Society all deserve thanks for the energies they put forth in producing our first volume. We hope that the final technical product has been both interesting and useful to you, the reader. Producing the first volume of IEEE Software has not been without its trials and tribulations; however, it has been an interesting, demanding, and rewarding experience, both technically and personally.

As with all things that grow at such a dynamic rate as we have (we grew from zero subscribers to over 21,000 in our first year), there are always some accompanying difficulties. We have not been able to respond to all authors in a timely manner concerning the disposition of their manuscripts. We are very sensitive to the issue of the timely processing of papers. Delays in reviewing are, in part, due to a heavily overworked refereeing community. I am very much interested in having as many software professionals as possible participate in the refereeing process. Would those of you who are interested in assisting with this worthwhile endeavor please send me your name, address (both physical and electronic), phone number, and the subject areas of the Computing Reviews Classification System in which you feel competent to review articles. Also, indicate how many articles per year you can review.

Growth is often accompanied by change, and this is true in the case of IEEE Software. In this second volume, we will publish three theme issues: one on Ada applications and environments; one on experiences with distributed systems; and one on operating systems for highly parallel environments.

Our first special issue, on Ada, will be in March 1985. I also hope to publish a few detailed interviews with distinguished scientists on topics of general interest sometime this year.

We are initiating a campaign to have as many manuscripts and reviews of articles transmitted to us electronically as possible. You can either submit six copies of your double-spaced, typewritten manuscript to me at the following address,

Bruce D. Shriver
Editor-in-Chief, IEEE Software
IBM T. J. Watson Research Center
PO Box 218
Yorktown Heights, NY 10598

or you can send me a single copy of the double-spaced, ASCII file output of your word processor to either of the following two electronic mail addresses:

(1) b.shriver@computerociety.compmail + service; or
(2) shriver.ytvmv@ibm-sj on CSNET.

Each department editor now has the appropriate compmail + address listed along with their physical addresses.

To remain as technically viable as we have been in our first volume, we need to continue to have submitted high quality technical articles on important issues to the software community. The problems that we continue to have in the design, implementation, use, and evolution of software systems (both in the application and system domains) are forcing us to reconsider the very way in which each of these processes is done.

Our existing tools are continually being changed as we learn more about what is done and what needs to be done in each process. In many cases, the tools must be redesigned, not only to accommodate new needs but also so they can be used with other tools to form an “integrated” tool
set. Although we have had some tools for building specific components for a number of years (for example, compiler compilers), we are only beginning to develop "meta-tools" for automating the construction of complex, integrated development environments.

By "tools," I do not simply mean entities such as specification languages, syntax-directed editors, and symbolic debuggers, but also the formal mechanisms that we employ as well. We are undertaking the slow, costly evolution from the current "arts and crafts" orientation of software development to one that is more scientifically based because we believe that a formal approach to the specification, design, prototyping, implementation, and verification processes is necessary if we are to improve each process, construct reliable and efficient software, and have higher software productivity. I do not argue for formalism's sake, but endorse formal methods that shed light on the real problems we must resolve. Therefore, an important part of an "integrated tool set" is its formal components. That is why I welcome articles that bridge the applied-formal tools gap.

There are often many different principles or methods for using tools. If we follow the principle of "reusability" when implementing a system, for example, we attempt to develop as many subsystems as we can so they can be "reused" in other parts of the specific system. If we can do this in such a way that the subsystems can be reused in other systems as well, either in a prototype or production context, we will have leveraged our efforts significantly. Can we generalize the principle of reusability to the subparts of the specification, design, prototype, validation, and documentation of a system as well? Can we develop methodologies (integrated groupings of formal and applied principles and methods) for building systems? Can methodologies span application domains—for example, are the methodologies for building expert systems the same as those for building real-time process control systems, or transaction-oriented commercial systems, or avionics communication systems? It would be useful for those of you who have had some experiences in this area to share them with the general readership.

If we are going to provide aids to help automate the software development process, we must understand not only the process, but also the entities that are produced. We must model both. Such modeling has been useful in providing a number of results, from the chief-programmer-team concept and code reviews in the "process" camp to the queueing models of operating systems usage and the data dependency graph models for optimizing compilers in the "entity" camp. The basic thesis is that in order to improve the present software development process and the resultant entities, we must first understand them. We can then propose changes to this process, to the formal and applied tools we employ, and to the entities. Then, after implementing these changes, we can evaluate them. "Software" behavior models may lead to methods to automatically reorganize programs for distributed and/or real-time applications. Some investigations in alternative software development environments are being undertaken, and we hope to report on them in our magazine.

One of the challenges for IEEE Software in 1985 and beyond will be to continue to provide a timely environment for the presentation of articles on these and related problems in the software domain. We welcome your participation in this endeavor.

Bruce D. Shriver
Editor-in-Chief