Preface

From the earliest days of computing, the need for effective software development tools and techniques has been evident. Today's vast backlog of applications represents unrealized significant savings in the application of computing because of delays in developing the software. This issue of the *IBM Systems Journal* presents tools and techniques that offer gains in applications development productivity while at the same time improving the quality of the product produced by programming and systems staffs.

In the current practice of producing software, separate tools and processes are used in the design phase and in the development phase. As a result, the interpretation and translation of design specifications into executable code leave room for redundancy and are subject to human error. Archibald, Leavenworth, and Power describe ADAPT, a new tool allowing for the integration of the design and development processes. ADAPT provides a single and consistent development language that can be used to support the system design and decomposition processes as well as the extension of component and module specifications into executable code.

An additional dimension is added to the integration of the systems design and programming processes by Pazel, Malhotra, and Markowitz in their paper, which describes the programming language EAS-E. EAS-E provides both procedural and nonprocedural support for the manipulation of data base and main storage entities. The language is implemented with the entity-attribute-set view that uniquely clarifies data structure presentation.

Without specific guidelines for program module design and decomposition, programmers may generate code that is difficult for others to debug and maintain. Rogers describes a hierarchical architecture for program design that has improved the maintainability of programs created at the IBM Toronto Laboratory. He describes a four-level hierarchy that can be applied to any program design, and which, in addition to improving maintainability, simplifies the design process, resulting in a shorter learning curve for new programmers.

An effective library system aids the program development process by serving as a central repository for source and object code, thus reducing the probability of the existence of multiple incongruous versions of the same program. The system described by Prager provides control of update capability through a "check-out" approach. In addition, the system offers such unique features as automatic module recompilation to ensure the integrity of text and source files.

Programs that automatically generate, execute, and evaluate test cases offer significant benefits but have seen limited practical use. Bird and Munoz describe the state of the art of automatic test case generation and checking. Their paper also describes the design and development of test case generation programs for the PL/I compiler, for a Sort, and for a Graphic Display Manager. In each case unique design problems were overcome in the development of these test case generators. The result of the use of automatic test case generation can be code that has been more fully exercised and therefore of a higher reliability and quality standard than could be achieved using manual testing procedures.

Although automated testing procedures offer great promise, manual test and debugging procedures will predominate for the foreseeable future. Maurer presents the unique features of an aid developed to support the development and testing of systems using the Interactive System Productivity Facility.

The design considerations presented should serve as a model for those developing and using similar test and debugging systems.

In many applications of computing, the reliability of the product produced by fallible human programmers must be assessed. Misra reviews the generally accepted methods of statistical software reliability prediction and presents a case study that illustrates the applicability of one of the techniques. Although the work is preliminary, the correlations with actual experience are impressive. Extensions of this work may be valuable in determining if the reliability of a software product will be sufficient to meet critical requirements, and to answer the question: "How much testing is enough?"

Given that the development process yields a system or program that functionally meets the user's requirements, the next critical hurdle is performance criteria. In spite of the vast price/performance gains experienced in our industry, the performance of a single module or system may be subject to critical inspection. Power presents a review of the design criteria for a program that can aid in the identification of performance problems: the execution analyzer. The author starts with a survey of the approaches used in the development of such tools and then illustrates his conclusions with specific examples from his own experience.

Even though the task of program and systems development is most certainly still properly characterized as an art, it is with the extension of such tools as described here that the art will come closer to being science.

The *Systems Journal* gratefully acknowledges the participants, contributors, and organizers of the IBM Programming Productivity Tools Symposium held in November, 1982, from which many of the papers presented in this issue have been derived.

John Lacy
Editor