Preface

This chapter of the twenty-fifth anniversary issue of the *IBM Journal of Research and Development* deals with computer science and software technology. When publication of the *Journal* began, the cost of software development was a small fraction of the total cost of developing a computer system. Now, the larger part of the development dollar is spent on software, and user programs add to this cost. Another significant investment is required to discover and extend the fundamental concepts on which practical programs are based.

Most of the papers in this chapter review IBM's involvement in bringing the relatively primitive aids to programmers and users of the 1950s to the elaborate operating systems, data base management systems, and language processors of today. Most of the authors have themselves been active in the field throughout this period and have contributed to the history. In addition to reviewing the extensive, if sometimes informal, literature, we have encouraged the authors to share their personal recollections with us, to re-assess some of their decisions of the past, and even to speculate a little about the future. The styles of the papers are quite varied, reflecting not only the approaches preferred by the individual authors, but the nature of the work reviewed as well.

Not all important IBM contributions to computer science are included, of course; there is clearly insufficient space to do that. And some of the papers deal more with technological achievements than fundamental contributions to computer science. This seems appropriate since IBM has been in an excellent position to implement and test experimentally many ideas that originated with researchers both inside and outside of the company. A good example of the magnitude and generality of products resulting from such efforts is IBM's operating systems. Two of the papers in this chapter deal directly with operating systems.

The first, by Auslander, Larkin, and Scherr, explores the emergence of an operating system discipline using MVS as an example. From beginnings as a small set of commonly required functions, operating systems have evolved to address today's expectations for easy, general purpose access to data processing power. In reviewing how IBM responded to emerging requirements, some perspective on the origins of functions now expected of all operating systems is provided.

The second paper discusses another operating system—VM/370. It is a member of a class of operating systems distinguished from other operating systems primarily by its approach to resource management, but by other characteristics as well. The evolution of VM/370 is reviewed by Creasy, leader of the group that, at what is now the IBM Cambridge Scientific Center, developed the experimental operating system on which VM/370 is based.

Memory management pervades operating system design and is an important consideration in the design of other software as well. Three long-time workers in this area, Belady, Parmelee, and Scalzi, collaborated on a paper that reviews the schemes devised throughout this period to relieve the application programmer of memory occupancy preplanning and to make practical the sharing of memory space among multiple programs.

Data management, once an important operating system function, is now often regarded as a separate technology. The evolution of data base technology in IBM is reviewed by McGee, an early contributor to this area. He demonstrates how our growing understanding of the nature of data influenced the design of programs, both inside and outside of IBM. This paper treats data structuring methods, data protection facilities, and high-level data languages, including the enduring report program generator.

To improve programmer productivity without significant sacrifices in computer efficiency, there has been a continuing trend throughout the past quarter century toward ever higher-level programming languages. Throughout this period IBM has contributed programming languages that allow quite diverse styles of programming and that serve the needs of a variety of subcommunities of programmers. One of the contributors to this history, Sammet, reviews this period of rapid programming language development.

Equally important are the language processors, which must produce efficient, machine executable code from the symbolic statements of the programmers. A key figure in this long-term effort, Allen, reviews assembler, compiler, and interpreter technology throughout this period. Her paper demonstrates that, although some fundamental problems of the computer scientists of 25 years ago have been largely solved, others are still with us. Specifically, the trend toward ever higher-level languages makes optimization of object code a continuing challenge.

In grappling with programming languages and their processors, it has long been recognized that formalisms could greatly facilitate communication among language designers and the developers of language processors, and such formalisms might also provide deeper insights into the languages themselves. A fundamental contribution in

this area is Backus-Naur Form—a notation for describing the syntax of programming languages. This is reviewed briefly by Sammet in her paper. Although no generally accepted method for formally defining the meaning of programming languages exists, a basic method has been developed at the IBM Laboratory Vienna, called the Vienna Definition Language (VDL). It has been applied to a large, complex language, PL/I. One of its developers, Lucas, reviews the method and compares it to successor approaches.

Throughout the history of data processing, workers have sought methods to model computing system performance, including that of its software support, so as to predict the performance of planned systems, to optimize the performance of existing systems, and to gain insights into complex system interdependencies. Two extensively published participants in this activity, Bard and Sauer, review the history, theory, and application of computer performance modeling, with particular emphasis on the contributions of IBM.

The general purpose programming support dealt with in most of the papers in this chapter only provides the soil in which application programs designed to solve specific problems can grow. IBM has participated rather directly in this aspect of computer science as well. As a sample of this activity, we include a paper by Flatt, who has spent much of his professional career dealing with large-scale scientific computations. His paper reviews the progress made in modeling complex energy- and environment-related phenomena, including studies of some of the consequences of energy consumption on the environment.

Editor