Preface

The themes of *quality* and *productivity* are currently receiving wide attention in both the technical and business press. The two topics are closely intertwined and are viewed by many as fundamental goals in all our business's plans.

The quest for quality and productivity improvements in large-systems software development is an important one to IBM and its customers. Such improvements are essential to sustain the growth in function we have come to expect from computing systems, in an environment where complexity and diversity of interconnection seem to be ever-expanding. The reliability, ease of learning, and ease of use of system software have become ever more critical considerations and place increasing demands on the software developer.

The universe of solutions to these problems is large and varied and continues to receive attention throughout the industry. New languages and support environments, very fast response times for developers, intelligent work stations, home terminals, and assistance from expert systems are only samples among a growing list of promising solutions.

There appear to be a number of factors that play a more dominant role in the large-systems software development process than in smaller-scale efforts. When projects begin to involve hundreds of developers, a critical focus must be placed on the areas of organization and structure, progression and quality assurance of intermediate work products, training of personnel, and supporting software tools. These areas are the focus of this issue of the *Systems Journal*. The first paper, by Humphrey, provides a perspective of the other articles which follow.

The paper by Radice, Roth, O'Hara, and Ciarfella takes a view of programming as a structured process and depicts an architecture to describe the working environment and flow of products through the process. With this structure in place, the authors propose that the appropriate software tools can be produced to effectively support the quality and productivity objectives. By contrast, it is thought that too often in the past the nature of the supporting tools determined the programming process.

The programming process architecture was the outgrowth of an internal IBM study described in the paper by Radice, Harding, Munnis, and Phillips. The investigators interviewed a number of development groups at different locations to identify programming tools, methodologies, and practices that proved to be effective. The study provides insight into programming development within IBM, and describes with some detail how proven methodologies can be identified and promulgated within a relatively large development community.

Hoffnagle and Beregi take a broad look at the opportunities for automating software development. By examining the developer's needs, as well as the shortcomings of current support systems in light of an evolved state of the art, they have developed requirements, direction, and an architecture for a new level of software development environment. They address such issues as portability, data sharing, and automated process control.

Teaching software engineering techniques through advanced education within IBM is described in the paper by Carpenter and Hallman. These authors review the needs that led to the creation of specialized curricula. They use a tutorial format to provide an understanding of the topics taught in the classes.

The use of the computer to aid in the collection, organization, analysis, and presentation of system requirements information is developed in the paper by Mays, Orzech, Ciarfella, and Phillips. The structure and consistency provided by the system is a notable example of an application supporting the program development process in which the data are primarily narrative, with the interactive access adding substantial value to the user.

The paper by Jones details efforts to analyze the cause of program defects with the goal of defect

prevention. Enhancements that incorporate the results of the analysis into an Entry-Task-Validation-Exit (ETVX) paradigm that is used within IBM are described.

Assessing progress in quality and productivity, as programming languages change, depends on having an applicable metric. Flaherty provides insight into the use of people and lines of code as ongoing metrics of programmer productivity, thus allowing progress comparisons to be made against a large historical base of productivity data.

The *IBM Systems Journal* has evolved gradually in content and form since the first issue was published almost twenty-five years ago. As the new Editor, I wish to thank the succession of talented individuals who have helped this journal maintain pace with our flourishing industry. My predecessor, John Lacy, made a special contribution to this evolution by bringing a new design to the *Journal*, including color graphics. I will do my best to carry on this heritage of quality as we continue to explore and develop new themes.

The spectrum of potential topics for publication in the *Journal* has evolved and grown substantially from the first issue in 1962 to encompass exciting developments in computer graphics, communications, and intelligent workstations. The users of IBM systems continue to expand applications of these machines into increasingly diverse and demanding areas, while the proximity and computing convenience provided by the Personal Computer have enabled this tool to augment the problem-solving process for a new generation of computer users. To do justice to the breadth of interests of our readers is challenging, and we continue to value your suggestions on topics and content. As always, we welcome inquiries from prospective authors.

Gary Gershon
Editor