Preface

The acceptance of software, from the mundane to the complex, depends fundamentally on the degree of quality evidenced by that software. Low-quality software is a burden to users and is eventually either discarded or, in the absence of alternatives, tolerated. High-quality software is accepted and promoted. Knowledge about software quality and the ability to practice it have been progressing slowly but inexorably throughout the industry.

This issue presents a broad view of progress in software quality, with a focus on IBM's worldwide efforts and results. There are eleven contributions, including an overview on software quality and discussions of process, methods, tools, case histories, management, and challenges for the future. We are indebted to A. J. Montenegro of IBM Canada Ltd. in Toronto, Ontario, for his noteworthy efforts in soliciting and reviewing the contributions and in planning and organizing the issue.

The first contribution is a tutorial by Kan, Basili, and Shapiro on software quality from the perspective of total quality management (TQM). Definitions are given, a brief history is shown, TQM and its philosophy are presented, and the specifics of TQM as applied to software development are discussed. Remaining challenges are offered as areas for further study and practice.

Fred Brooks and his work on mythology versus reality in software development have become part of our collective understanding of the software arena. His work on the notion of a "silver bullet" for the essential complexity of software development is well known. Much of what has been discovered about software seems to impede progress toward improved software quality. In the second paper, Mays proposes and explains possible means for forging a silver bullet, based on reducing the inherent difficulties seen by Brooks in his essential attributes of software. The collected means include such promising techniques as design reabstraction.

It has long been understood that quality in complex software cannot be achieved without attention to the software development process. Billings et al. provide a case study from a project that has earned the highest rating in the Software Engineering Institute's Capability Maturity Model: the Space Shuttle Onboard Software project. The focus of the paper is the process and the growth in process maturity over time, building on innovative ideas, practical experience, and the lessons of trial and error.

Software quality is also affected by the management system under which it is developed. That management system has a number of aspects, including customer satisfaction, product quality, continuous process improvement, and the people involved. Kan et al. present a second case study, on management systems, drawn from the development of Application System/400* (AS/400*). The IBM Rochester site received the 1990 Malcolm Baldrige National Quality Award for these efforts.

One of the sophisticated tools that has been developed from new thinking about software quality is Cleanroom software engineering with its related processes, methods, and tools. Hausler, Linger, and Trammell describe the technological and mathematical underpinnings of this approach and show a staged implementation for Cleanroom software development. They also provide results of its use on one significant project in IBM and summary results for 17 other projects, totaling about one million lines of code.

One of the normal, expected events in the life of a software product is the modification of code written by someone else. It is also the source of much frustration and many errors due to a lack of understanding of the original author's techniques and design. Resolving this problem effectively would significantly aid in the immediate and long-term quality of software. O'Hare and Troan present their work on a system called RE-Analyzer and its related tools, which provide automated reverse engineering in a computer-aided software engineering (CASE) milieu. The technologies they bring to bear on this significant problem include reabstraction of source code, control partitioning for managing complexity, and structured analysis of constructed diagrams of data flow and state transitions and of models of entity-relationship data.

Capper et al. consider the impact of object-oriented methods on software quality. They provide three case studies that show how and when to use object orientation to improve software quality by taking advantage of reuse and code modularity. Defect rates for the three case studies demonstrate the effect on quality of the combination of object-oriented methods with standard software processes and illustrate the resulting ability to add function to existing software with high quality.

Looking at some of the latest results in software development and their impact on quality, Yakhnis, Farrell, and Shultz present possibilities for deriving programs from their specifications through generic algorithms, which are explained at length in the paper. This approach offers the opportunity to use formal methods for software development, with their attendant high quality, while avoiding the mathematical difficulty of formal proofs done by people. Since each generic algorithm can be used for a large class of potential programs, the effect of high quality in the algorithms is multiplied and reflected in the high quality of a number of resulting products.

Bhandari et al. provide examples and discussion on how defect data, retrieved during software development, can be effectively interpreted and used to correct the process and the product. The authors present the attribute focusing method, which is an extension to earlier work on orthogonal defect classification. The paper shows broad classes of projects that can benefit from this method, and the paper can be used to learn how the method is applied.

The next contribution to the discussion of software quality is included in the Technical Forum, a new section that was introduced in the last issue. The subject is IBM's current programming principles and practices, resulting from the efforts and experiences in improving software quality during the last few years, as presented by Bencher.

In a technical note, Watkins adds new insights to the discussion of reliability models for software development. He builds on work by Kan, published three years ago in these pages, by focusing on the variations of approach that are possible in early stages of modeling.

*The next issue of the **Journal*** will be a special issue on enterprise-wide database management and access as envisaged by IBM's Information Warehouse* framework and related work.

Gene F. Hoffnagle
Editor

*Trademark or registered trademark of International Business Machines Corporation.