Preface

Total sales of software worldwide exceed $200 billion annually, and it has been estimated that software errors cost the United States economy $59.5 billion, or .6 percent of the gross domestic product, each year. Despite this, the practice of mature software engineering is still, in many ways, a goal rather than a current reality.

Many attempts have been made to bring more rigor, predictability, and efficiency to software development. Some of these involve the capture of information relating to the requirements, architecture, and implementation of software systems. Others focus on the reuse of code and the design of tools for software development and testing. Model-driven software development (MDSD), the topic of this issue of the *IBM Systems Journal*, is an emerging technology that offers hope in introducing significant efficiencies and rigor to the theory and practice of software development.

While the use of models in design and testing is not new, MDSD is innovative in offering cohesive and comprehensive technologies for the use of models as the basis for all stages of the software development life cycle, including the full implementation, testing, and deployment of complex systems. This methodology also facilitates the automatic generation of significant amounts of code which otherwise would need to be generated manually. The papers in this issue of the *Journal* focus on MDSD's role in software engineering, describe several domain-specific implementations of MDSD-based projects, and supply an introduction to the standards, processes, and methodologies of MDSD.

In "Model-driven development: The good, the bad, and the ugly," Hailpern and Tarr analyze how and whether tools incorporating model-driven development can be used to meet the challenges presented by the complexity of today's products, their shortened development cycles, and higher customer expectations of quality. Brown, Iyengar, and Johnston, in "A Rational approach to model-driven development," explain how the development process can be facilitated by the portfolio of IBM Rational* tools, which have supported model-driven approaches for over a decade.

In "Architectural thinking and modeling with the Architects' Workbench," Abrams et al. present the design of and innovations embodied in the Architects' Workbench (AWB). AWB is a prototype tool facilitating the collection and organization of all of a system's architectural information (both structured and unstructured) and the implementation of the system. Sinha, Williams, and Santhanam, in "A measurement framework for evaluating model-based test generation tools," present a framework for formulating the metrics of complexity, ease of learning, effectiveness, efficiency, and scalability of these tools, and describe a case study. Aizenbud-Reshef et al., in "Model traceability," show how model-driven development provides new opportunities for establishing and using information on traceability. The establishment and use of information on traceability is achieved by defining and maintaining relationships between artifacts involved in the software-engineering life cycle.

Batory, in "Multilevel models in model-driven engineering, product lines, and metaprogramming,"

0018-8670/06/$5.00

introduces a multilevel paradigm of program development based on the confluence of model-driven development, product lines (i.e., creating a family of related programs) and metaprogramming (viewing programming as a computation). This paradigm clarifies the concepts of MDSD. The paper "Model-driven development: Assets and reuse," by Larsen focuses on the identification, organization, and reuse of reusable models.

A tool for model-driven development is presented by Leroux, Nally, and Hussey, in "Rational Software Architect: A tool for domain-specific modeling." Rational Software Architect provides a powerful capability for integrating domain-specific languages with UML** (the Unified Modeling Language**) in a single toolset. Balmelli et al., in "Model-driven systems development," apply model-driven design principles to the development of systems. The approach presented in this paper uses the RUP* SE (Rational Unified Process* for Systems Engineering) architecture framework and extends traditional systems engineering methods in order to adapt to a systems development environment characterized by rapidly changing conditions and requirements. Chowdhary et al., in "Model Driven Development for Business Performance Management," show how the model-driven design methodology and framework was used to create a Business Performance Management solution for monitoring in the Distributed Enterprise Services (DES) application.

Model-related standards are described by Selic in "UML 2: A model-driven development tool," and by Czarnecki and Helsen in "Feature-based survey of model transformation approaches." The former paper explains the reasons for revising UML to better support MDSD tools and methods and overcome some of the resistance to MDSD methods. The latter paper proposes a taxonomy for the classification of several existing and proposed approaches for the transformation of models into other models.

Finally, Chandra et al., in the Technical Forum paper "Using logical data models for understanding and transforming legacy business applications," address the challenges of analyzing and transforming legacy business applications, focusing on mainframe-based systems written in COBOL (common business oriented language).

We would like to thank Brent Hailpern and Peri Tarr for their efforts in the conception and coordination of all aspects of this issue.

The next issue of the *Journal* focuses on collaborative computing.

David I. Seidman, Associate Editor

John J. Ritsko, Editor-in-Chief