Preface

The creation of new technology for software development and rapid technology transfer from research into product development remain notable challenges for software researchers and practitioners. One model for cooperation between universities, governments, and companies is the applied research center. This model was reformulated for software and implemented in 1990 at the IBM Toronto Software Solutions Laboratory in Ontario, Canada. Called the Centre for Advanced Studies (CAS), it brings together the efforts of researchers and developers under a funding program supported by a number of North American universities, the Canadian government, and IBM.

This issue presents some of the work sponsored by CAS. An introductory essay describes the concepts and operation of CAS, and six papers are drawn from current projects. We are indebted to J. Slonim, Head of Research for CAS, IBM Canada Ltd., in North York, Ontario, Canada, for his solicitation of these papers and his coordination and development of the issue.

The model for a Centre for Advanced Studies as envisioned by IBM is an extrapolation of earlier models for applied research involving researchers and practitioners. In the case of CAS, it has been reformulated for the needs of software and the software community. CAS required a rethinking of such factors as time from idea to product, utilization of researchers for solving problems found in exploiting new technology, use of prototypes, cooperation and communication between universities and IBM, and the policies of funding sources. In an introductory essay, Slonim et al. discuss why this model was developed; what resulted from that rethinking of factors; what principles, processes, and initiatives are identified with CAS; and what the impact has been so far.

In the first paper on results from CAS projects, Bauer et al. present an architecture for software application development in the context of heterogeneous systems and distributed execution. The architecture deals with concerns such as application environment, legacy applications, new applications, data access, software management, visualization, testing, and transparency. The authors argue for a peer-to-peer view of such systems and describe their prototyping efforts to date. This work is known by the acronym CORDS, which stands for COnsortium for Research on Distributed Systems.

Continuing the story of CORDS from the previous paper, Bauer and a different group of coauthors focus on a reference architecture for management of complex distributed computing systems. They decompose the management problem into three areas of system behavior—network, operating system, and application—and demonstrate how these areas are interdependent. A detailed example of a hospital network is used to present five prototype implementations examined during the course of the project.

Another project has examined the use of predicate-based software testing strategies, with BOR (BOolean operatoR) testing as the strategy under comparative study and compound predicates as the focus. Tai et al. give the results of empirical tests of BOR and other strategies. They conclude that BOR is practical and effective for specification- and program-based test generation. Their test results are preceded by a description of BOR and the other strategies in the study.

Consens et al. describe their approach to visual display and manipulation of complex databases through their Hy^+ generic visualization system and its support of the visual query language GraphLog. Examples are presented for software engineering and network management. In addition, the architecture and design of Hy^+ and its

query processing and graph layout components are shown. They conclude that Hy^+ provides an ease of application to a wide variety of fields where visualization might apply.

Program understanding is an area of software engineering that is currently under extensive study as a means to recreate the design of existing software when that knowledge has not been maintained or is lost. Buss et al. formed a multi-university team to examine a large legacy system using various reverse engineering methods and to compare the results. The methods studied were defect filtering, structural redocumentation, and pattern matching. In order to facilitate their own work and to make all these tools available in an integrated environment, they developed a tool integration scheme, which they also describe.

Heineman et al. show the results of their work on technologies for development of quality software as seen from the perspective of the software development and maintenance processes. They view these processes as having a life cycle that encompasses design, instantiation, use, and improvement. The paper contains their goal-by-goal analysis of the range of process techniques under study, utilizing the goal structure in the referenced work by Curtis, Kellner, and Over. They draw conclusions about the span of research into goals for processes and construct a hierarchy of objectives that includes process management, improvement, automation, and understanding.

We are pleased to announce that information about the *IBM Systems Journal* is now available electronically to IBM customers through IBMLink. By entering INEWS SYSJRNAL on the command line of the IBMLink main menu, customers have direct access to information about recent issues of the *Journal*, ordering of subscriptions and single issues, and such matters as letters to the Editor. (IBM employees view this same information by entering INEWS SYSJRNAL, if the SYSJRNAL category is installed on their system.)

The next issue of the ***Journal*** will be an index covering 1962 through 1994, consisting of three parts: an author index, a subject index, and complete abstracts by issue. It will also contain updated author guidelines and a review of suggested references for writing technical papers.

Gene F. Hoffnagle
Editor