evolutions bring change — often change for the better. Computer science has witnessed many revolutions, including the emergence of high-level programming languages, database-management systems, and artificial intelligence. Each in its own way demanded changes in how software is developed. Now, the advent of interactive systems with human-computer interfaces is causing a revolution that demands new approaches to developing software of a different sort.

Building any software system is a complex task, even when the system is computation-intensive and does not emphasize the interface. Injecting the ever-changing, always-unsatisfied human into the process necessitates a quality human-computer interface, complicating an already complicated task. An interactive system — one with a human-computer interface — is not judged solely on its ability to compute. It is also judged on its ability to communicate. In fact, if users cannot communicate effectively with an interactive system, its computational ability may be inaccessible.

Developing an interface is very different from developing conventional software. The traditional methods, techniques, and tools that work well for software development are proving not to work so well for interface software. How many software-development methodologies specifically include procedures for producing the interface? New technology is constantly increasing the developers’ palette of interaction devices, styles, and techniques, yet how many conventional languages have constructs for incorporating a window, an icon, or a mouse?

We must support the revolution in user interfaces with new methods, techniques, and tools. Our goal, of course, is better interfaces.

Articles. The articles in this issue address the revolution’s most important areas, from both a theoretical and pragmatic perspective. Virtually all the articles address what is perhaps the most far-reaching area of the revolution: the advent of interactive systems for developing the interface. Thus this issue gives a multifaceted look at approaches to developing interface software of a different sort:

- The first article, by Brad Myers, is an overview of interface-development systems, which he calls user-interface devel-
Window on the Computer

opment systems. Myers classifies UIDS by how they let the programmer specify the interfaces and gives examples of each type. His article has the foundation for the tool-related articles that follow it.

* One of the newest user-interface management systems, the User-Interface Development Environment, is described by its creators, Jim Foley, Mark Chisholm, Srdjan Kovacevic, and Kevin Murray.

UIDE is built around a knowledge-based representation of the conceptual design of a user interface. UIDE supports design at a high level of abstraction: It generates a high-level specification of the interface from information provided by the designer. By using the same knowledge base, the interface developer can generate a new interface design with the same functionality as the original design. This lets users try many functionally equivalent interfaces for the same application.

* Gerhard Fischer presents another knowledge-based approach to enhance and support communication between humans and computers. His system architecture has an explicit communication channel between the human and the hardware and an implicit communication channel between the human's knowledge base and the computer's world knowledge. Fischer uses two intelligent support systems, Framer and Crack, to illustrate this concept. He concludes with a pragmatic description of lessons learned and challenges ahead.

* Matthew Hodge, Russell Samet, and Mark Ackerman describe an interface system, Athena Muse. Muse is an experimental kit for the construction of multimedia learning environments. Learning environments developed with Muse offer a diverse set of complementary interaction techniques, styles, and devices. An interface developer can choose from four representation approaches: directed graphs, multidimensional spatial frameworks, declarative constraints, and procedural languages.

* David Rask, Michelle Lund, and Henry Ramsey contribute a retrospective on one of the oldest UIIMS projects. They describe building a commercial UIIMS from the perspective of the members of the development team. While the OASIS application domain is generally CAD/CAM/CADL, users can extend its functionality by using a command
Another research area in interface design is communication issues, especially how application semantics fit into design of the interface. Developers advocate separation of the interface from the application, but where and how to achieve that separation is not easy to determine, especially for direct-manipulation, highly interactive systems. Two articles address this problem:

- First, Rex Hartson addresses how to separate components in an interactive system, illustrating and comparing approaches to separation. He then describes how to achieve control and communication among those components. Hartson presents the concept of two communication levels to support separation.

- Next, Dave Hurley and John Sibert describe how to model the interface between the user interface and the application. Their conceptual modeling system contains constructs and abstraction mechanisms that structure knowledge in a model and primitives that give behavior to knowledge in a model. They use the interaction semantics of a spreadsheet to illustrate the practical application of their system.

- Once we learn how to produce a user interface—especially by using interactive tools—what do we do with it? The final product, the interface itself, must then be evaluated. John Thomas and Wendy Kellogg explore an ecological gap between what is observed during interface evaluation in a laboratory and what happens when the interface is used in the real world. Thomas and Kellogg look at the ecological gap as it relates to users, tasks, systems, and the real-world context. They propose specific techniques to be used at different phases of development for evaluation and tuning the interfaces in a real-world context.

This article is a nice closing for this special issue because it points to the need to do more empirical work on the human-computer interface development process. It is particularly appropriate for this publication; publishing such articles primarily in human factors journals propagates the very gap Thomas and Kellogg address!

Acknowledgments

I thank Ted Lewis for his encouragement and patience in guiding me through the task of editing this special issue, and June Martin of his staff. Thanks also to those who provided timely and thoughtful reviews; your efforts vastly improved the quality of the final product.

This issue would not have happened without Joanne Lee Bogner, my secretary, who kept track of the mountain of details, kept authors (and me) on schedule, handled tons of correspondence, found answers to unanswerable questions, and on and on. Rex Hartson and Bob Schulman gave much moral support and encouragement. Here it is, guys!

Deborah Hix is a research computer scientist on the faculty at Virginia Polytechnic and State University in Blacksburg. She is a principal investigator on the Dialogue Management Project. This project is concerned with how to achieve quality human-computer interfaces through development of concepts for human-computer interface management and through development of specialized methodologies, techniques, and tools for producing the human-computer interface.

Hix received a BS in mathematics from Emory University and an MS in information science and a PhD in computer science and applications from Virginia Polytechnic Institute and State University. She is a member of IEEE, ACM, and the Human Factors Society.

Address questions about this issue to Hix at Virginia Polytechnic Institute and State University, Computer Science Dept., 562 McBryde Hall, Blacksburg, VA 24061.