*The focus of CASE research and development has shifted from making sure each tool works to making sure all tools can work together. Major vendors are striving to balance comprehensiveness and compatibility.*
When IEEE Software last devoted an issue to computer-aided software engineering (March 1988), a primary concern was making sure that each CASE tool worked. Back then, the major CASE players were mainly small software firms, universities, and research laboratories.

Things are different now. Several prominent software and hardware companies have announced comprehensive CASE strategies and frameworks, and several domestic and international CASE standards efforts are underway.

In the last four years, the software industry has realized that no one size fits all the shoemaker’s children. So the focus of CASE research and development is shifting to ensure that tools from different vendors can work together. Integration is becoming a key issue in developing and deploying CASE tools.

The technical and managerial difficulties we face in integrating CASE tools are rooted in the history of CASE technology, as summarized in the time line in Figure 1.

**EVOLUTION OF CASE**

The tools and methods that have evolved into today’s CASE environments were designed to support application development. Therefore, CASE has been significantly influenced by the characteristics of the applications being built at the time and the methods that existed for building these applications.

New applications drive the creation of new systems-development methods. Because these new de-
Development methods are complex, tools are developed to help use them. These more powerful tools, in turn, make it easier to develop new applications and methods.

**Applications.** In the 1970s, most business applications were batch transaction-processing systems written in third-generation languages. As database technology matured, more complicated, data-intensive, on-line transaction-processing systems were developed. In the late 1970s, several decision-support systems that help users analyze data via interaction with decision models were built.

In the 1980s, Ada was created when the use of real-time software embedded in control-and-communication devices grew. In the mid 1980s, expert systems and knowledge-based applications received a lot of attention. And in the late 1980s, organizations began to use strategic information systems to stay competitive, and top managers began to use executive information systems to retrieve internal and external information through desktop workstations.

In the 1990s, these information systems must be integrated to span all business functions, organizational levels, and global locations. Building these systems will require combining several enabling technologies. The pressure to build these systems on time and the complexity of the emerging enabling technologies (like client-server architectures and graphical user interfaces) are the major driving forces behind the push for integrated CASE.

**Methods.** Structured programming was one of the first efforts to develop a systematic method to support systems development. Because of the high cost of correcting errors introduced early in the life cycle, structured techniques for design, analysis, and planning were subsequently developed.

In the mid to late 1980s, real-time systems design and object-oriented analysis and design methods were developed to amend the shortcomings of structured techniques. This period also witnessed the development of some interrelated methods and techniques (like information engineering) that took into account the entire life cycle. The rigor and complexity of these methods dictated the use of CASE to apply them.

Because code generators and fourth-generation languages have simplified downstream activities somewhat, the development bottleneck has shifted to upstream activities, including systems planning, enterprise modeling, and requirements engineering.

The quality of upstream products is determined by how well systems personnel can get users and managers involved in development. Participatory design methods like Joint Application Design have been particularly helpful in improving user involvement in design. They complement other methods and have become well accepted in the last five years.

Structured methods, developed in the 1970s, became popular in the 1980s, fueled by the rise of graphical notations and end-user orientation. However, many structured methods deal with only one or a few aspects of information-systems models, and critics say they are too imprecise and ambiguous.

Formal methods are more rigorous, but they are often not appropriate for communication between users and developers. Formal methods and structured methods have complementary strengths and weaknesses; the possibility of integrating them is attractive.

Objective measures of productivity
and quality are essential if we are to improve the development process. Measurement can be improved when process management is fully integrated in CASE environments because metrics can then be collected automatically. The field of software metrics will continue to grow in the 1990s, because using metrics in conjunction with tools and methods and applying statistical process control will help us better manage the development process.

Many current CASE tools, which were designed to support specific methods, make it practical to apply methods. The symbiosis of CASE tools with methods suggests that advancement in integrated CASE may rely on the development of tools that integrate many modeling perspectives within and across all life-cycle phases and across many application domains.

**Tools.** Systems development generates volumes of complicated development information that must be captured and analyzed. A CASE environment lets systems developers "document and model an information system from its initial user requirements through design and implementation and lets them apply tests for consistency, completeness, and conformance to standards." The use of CASE technologies is becoming a critical factor in the success of large-scale systems projects.

In the early 1970s, first-generation CASE tools (such as PSL/PSA) were generally mainframe and text based. Interest in these tools helped spark the development of structured methods, which led to the realization that automated tools were necessary to gather and help analyze the voluminous development information that the use of structured methods generated.

The advent of graphical user interfaces on PCs and workstations led to graphical, front-end tools to support structured methods. However, the information captured by these early tools was stored in the tools themselves and was generally not transferable among tools.

Second-generation CASE tools developed in the early 1980s were designed mainly to support structured methods that use graphical notations, such as dataflow diagrams for structured analysis and structure charts for structured design. They incorporate better analysis functions to enforce the method's rules.

The detailed development information these tools capture is stored in a project dictionary to be shared by other CASE tools in the same environment. However, this kind of integration is limited to tools from the same vendor and, generally, data within the same project. It is possible to loosely integrate a CASE environment with add-on utilities that import and export development information and from and to the project dictionary. In a few cases, it is even possible to link tools sold by different vendors if there are agreed-on data formats or proprietary application-program interfaces.

The repository-based CASE products that became available in the late 1980s offer enterprise-wide and project-level local repositories. The repository integrates a set of tool kits for planning, analysis, design, programming, testing, and maintenance. However, many of these products depend heavily on the method used and tend to support only certain types of application development.

The 1990s will be an era of systems integration through open systems, driven by the demand for ever faster development of highly integrated, very complicated strategic information systems. Integrated CASE environments empower IS organizations to deliver these systems on time and to migrate systems among open-systems platforms.

To meet these demands, an integrated CASE environment must be based on a flexible framework that provides a cost-effective tool integration mechanism, encourages portable tools, facilitates the exchange of development information, adapts to future methods, and extends to other engineering disciplines.

---

**IN THIS ISSUE**

The eight articles in this theme issue address various aspects of CASE integration.

In the first article, we introduce a technical framework for an integrated CASE environment based on the reference model developed by the National Institute of Standards and Technology and the European Computer Manufacturers Association and propose an organizational framework that places the technology in the context of information systems development and management at the enterprise, project, and team and individual levels.

In the second article, Alan Brown and John McDermid examine the differences between integrated project-support environments and CASE. The authors outline IPSE's shortcomings and advocate integration at higher levels — to improve productivity and quality.

Next, Ian Thomas and Brian Nejmez define what we mean by tool integration in software development because current definitions are not precise enough. They define tool integration by identifying its relationships with other tools in the environment and the properties that characterize those relationships.

The last five articles make up a set of experience reports. Christer Fernström, Kjell-Håkan Närfelt, and Lennart Ohlsson review the principles, architecture, and experiments developed as part of the Eureka Software Factory project, a 10-year effort by a consortium of 13 European partners. ESF takes a market-driven approach to CASE integration and has adopted a communication-centered architecture to accommodate different types of tools from different producers.

Then Peiwei Mi and W. Scacchi describe their process-centered approach, which they used to integrate tools in their Softman development environment.
key components of their approach are software process models, a process driver, and separate developer and manager interfaces.

Database-intensive systems are addressed by the team that developed DAIDA, an integrated development environment created as part of the European Strategic Program for Research in Information Technology. Reported by Matthias Jarke, DAIDA's key strategies for integrating CASE are concept modeling, process modeling, and knowledge-based assistants to map requirements to designs and designs to programs.

Another architecture, this one based on hypertext technology, is reported by Jacob Czyński and Karl Reve. HyperCASE integrates tools by combining a hypertext-based user interface with a common knowledge-based document repository. Integrating CASE and hypertext to improve user interfaces is a promising approach.

Finally, Alan Hevner, Shirley Becker, and Lenard Pedowitz propose an integrated CASE environment to support Cleanroom systems development, one of the most successful formal approaches to systems engineering. The authors propose a repository-based supporting CASE architecture.

As we enter the second decade of CASE products that run on graphical workstations to support development by teams, our demands on CASE technology will continue to grow. Each of us must continue to work toward the adoption and successful institutionalization of integrated CASE in this decade. This issue of IEEE Software is dedicated to achieving that goal.

SIGNPOSTS AND LANDMARKS: A CASE READING LIST

In looking for key papers and articles that provide technical background on CASE and integrated CASE, it is important to realize that the literature includes several overlapping topics. To get a reasonably complete picture, your keyword search should include the terms CASE, software-development environment (SDE), software-engineering environment (SEE), and integrated programming-support environment (IPSE).

The principal CASE literature begins with the publication of papers on PSL/PSA (from the University of Michigan) and SREM (from TRW) in the landmark issue of IEEE Transactions on Software Engineering (January 1977) on requirements engineering.

Many important papers were collected and published in Tutorial: Software Development Environments, edited by Anthony Wasserman (IEEE Computer Society Press, Los Alamitos, California, 1981). This book includes papers about the goals of environments, programming systems, operating systems, and tool kits (including Unix); support for development methods; and issues of organizational support and human factors.


Also in 1988, Maria Penedo and Bill Riddle edited an issue of IEEE Transactions on Software Engineering on software-engineering-environment architectures (June 1988). This issue updated the developments on architectural alternatives and interface approaches, including an overview on the Ada Common APSE Interface Set by Patricia Oberndorf.

— Elliot J. Chikofsky, Associate Editor-in-Chief

Ronald J. Norman is an associate professor of information and decision systems at San Diego State University. His research interests are CASE technology, technology transfer and organizational change, and automated and object-oriented systems analysis and design.

Norman received a PhD in management information systems and organizational behavior from the University of Arizona.

Minder Chen is an assistant professor in the Decision Sciences and Management Information Systems Department at George Mason University. His research interests are CASE, methodology engineering, collaboration technologies, and expert systems.

Chen received a BS in electrical engineering from National Taiwan University, an MBA from National Chiao Tung University, and a PhD in management information systems from the University of Arizona.

Address questions about this article to Chen, Decision Sciences and MIS Dept., George Mason University, Fairfax, VA 22030; Internet mailchen@gmunix.gmu.edu.