A Self-Assessment by the Software Engineering Community

Software is increasingly identified as a critical constraint in the realization of all types of industrial and commercial systems. The findings of the last International Workshop on Computer-Aided Software Engineering (IWCASE), summarized here with a fair degree of editorial license, represent a cross section of opinions and perspectives from over 200 experts in software development technology from the academic, supplier and user communities. In effect, it constitutes one self-assessment of how well our profession is meeting the challenge of building a respectable engineering discipline, and the design automation tools required to support it.

IWCASE defines CASE in the broadest terms, namely tools and methods to support an engineering approach to software development at all stages of the process. By "engineering approach" we mean a well-defined, coordinated and repeatable activity with widely accepted representations, design rules and standards of quality. Tools that support such a software-engineering discipline are, by our definition, CASE tools, regardless of the specific phase, task or notation.

CASE has been successful in focusing attention on the need to establish software development as an engineering discipline. By automating many of the more routine software development tasks and performing automatic transformations between representations, CASE has (under the right circumstances) demonstrated an ability to boost productivity and prevent defects. Advanced tools are making it more feasible to introduce semiformal and formal methods to the development process by removing clerical overhead, enforcing rigorous design rule checking and exploiting expert systems technology to guide the specification process. Some CASE tools can enhance team efforts through coordination technology, project-wide consistency checking and shared design data. In some domains CASE Tools help *unlinearize* the software development process, making it more interactive and consistent with the way people really think and work.

But CASE is no panacea. We have only just embarked on a quest for excellence in software development. Many problems remain and debates still rage over which approaches are the most effective. In this context, the IWCASE findings on CASE become one yardstick by which we can gauge our present status as a profession, and a baseline for evaluating future progress.

Among the greatest challenges ahead is the need for tighter integration among tools in a manner that supports openness to a variety of methods, notations, processes, tools, and platforms. Understanding the software development process and getting developers to use software-engineering techniques correctly and consistently will remain a problem, especially in the face of evolving technology. A more disciplined approach to software process engineering is needed employing metrics, feedback and continuous improvements. Management will have to take a long-range economic perspective, modify objectives and incentives, and incorporate principles of Total Quality Management to ensure the ultimate success of CASE.

Underlying Themes

Threaded throughout many of the conference results were the themes of software quality, software development process, and management of expectations.

Quality is often cited as a primary objective for the pursuit of CASE. An essential concept behind the search for software quality is the replacement of defect detection and elimination (the traditional approach) with defect prevention. Prevention begins with better ways to capture, represent and validate the objectives and requirements of systems we are trying to build, and continues with verifiable transformations and refinements on the way to physical realization.

While CASE has already achieved substantial success in defect prevention, we are reaching a plateau due to the limits of our knowledge about the software development process. We need further research, as well as better metrics and metric collection facilities, to guide us toward continued improvement in the future.

But continuous improvement is based on the premise of a well-defined software development process. There is still much work to be done in defining generic processes, and domain-specific variants, understanding how they relate to product quality, and discovering how and why they break down in actual practice. Areas that are particularly weak in process definition are requirements elicitation, software maintenance, reengineering, and object-oriented techniques.

Management attitudes and expectations with respect to software development must change to match the reality. Software development is becoming a more complex and capital-intensive business. Information and the software that manipulates it are key strategic resources. To remain competitive, managers must acknowledge that software activity requires a major long-term commitment to technology insertion, human resource development and organizational change, with a commensurate allocation of capital.

Existing Systems

There is general agreement that tools are not effective without a supporting method to guide their use. Currently, CASE for software maintenance and reengineering is being hampered by the lack of a defined process. It is not surprising, therefore, that available tools address only a portion of the maintenance activity and are not well integrated with tools for new development.

All indications are that the design recovery process will involve incremental program understanding for the foreseeable future. Reverse engineering tools must be highly interactive and facilitate the capture of information with storage in a shared database or CASE repository. Where software is being salvaged for reuse, we need better ways to capture explicit information about business rules and policies; technology models; application form, fit, and function; and design intentions, history and rationale.

A class of support that has been largely overlooked is decision support tools for software portfolio analysis, build/buy analysis, maintain/rebuild analysis, and reuse/start-from-scratch analysis. The generally accepted economic models associated with software maintenance and evolution may have to be replaced with new models that recognize the longevity of applications, the inevitability of externally driven application modifications, the value of reusable assets, the true cost of new development, the opportunity costs related to time-to-market, and the impact of CASE technology.

Technology Transfer

A key challenge for CASE continues to be the management of technology insertion. Success with CASE depends critically on planning, managing expectations and early experiences with the technology. While there is no way to guarantee the initial success of CASE, experienced users cite adequate planning, preparation, training, capitalization, and executive involvement as factors that greatly increase the probability of an acceptable outcome. Common causes of failure include a short-term, "silver-bullet" attitude on the part of management, inadequate infrastructure (including planning process, training and tools), undercapitalization, inability to share a vision at all levels of the organization, and failure to match methods and tools to the organization's current level of maturity. Success with CASE depends partly on management's willingness to accept a long time horizon to evaluate payback, and a global (strategic and company-wide) perspective on CASE benefits. Early "failures" must be viewed as learning experiences

rather than a condemnation of the technology or individuals. In general, economic models need to be revised so they lead to correct investment conclusions, especially in the areas of software maintenance, reengineering and reuse.

Technology insertion could be aided by CASE tools that are scalable in their functionality and complexity (as well as cost), with a smooth transition path to more sophisticated usage. The potential for tools to aid in the implementation of CASE itself has not yet been adequately explored.

Technology transfer from research to commercial enterprise could be improved by developing explicit transition process models, realistic expectations, and better role definitions for those involved. A cooperative model appears to be more appropriate and more effective than one-way transition models. Better documentation of both technology insertion and technology transfer are needed to refine process and economic models and to support claims of long-term payback.

Effective widespread deployment of CASE will require greater emphasis on software engineering in university and commercial training curricula. This means focusing on generalized problem-solving skills, as well as software-specific formal and informal methods, requirements elicitation, architectural (i.e., system-level) design principles, team coordination and management, process engineering, and quality management techniques. Instructors should encourage the recognition of, and adherence to, engineering standards and disciplines, and strive to shift predilections toward reuse and system-level design. In general, commercial training should set more realistic expectations for CASE and provide a better balance between potential benefits and pitfalls.

Group and Process Management

A major dimension of the software challenge is scaling up for very large projects requiring many software engineers. To significantly improve coordination and overcome the overhead costs associated with interpersonal communication, we need better specification languages, better understanding of processes, and advanced coordination technology—or so-called groupware. Further research is needed on how to capture and formalize requirements and specifications. We also need research on how to organize and describe the development process, its own internal information requirements, interactions and products.

We have little agreement on basic definitions of quality and productivity or metrics that could indicate progress in these areas. Automatic collection of metrics on software artifacts as well as developer behaviors and interactions could provide raw data useful for gaining a better understanding of software development.

Coordination technology requires an underlying capability including fast, reliable multimedia communication over heterogeneous networks, and broadcast messaging. Beyond the current email tools, we need comprehensive task, defect and change management, multimedia electronic conferencing, and learning support for new team members.

Enabling Technologies

It is becoming apparent that a single design method will not adequately address all application domains, or be effective for maintenance and reengineering as well as new development. Also, differences in skill levels, styles, attitudes, cultures, goals, and constraints demand highly tailorable CASE tools. The goal is a technology that can accommodate many methods, notations, styles, and levels of sophistication, while supporting a rigorous software-engineering process. Both the techniques for describing and analyzing various software design methods, and the tool technology to accommodate multiple methods, are subjects of growing interest. Collectively referred to as metaCASE, the study and support of tailorable methods promises to improve the acceptance and effectiveness of CASE in the future by "meeting developers where they are" and by enabling more specialized approaches for specific application domains. MetaCASE capabilities can be achieved through modifiable metamodels to describe methods and drive tool behavior. Tools should also be flexible on a personal level—customizable to user style preferences and scalable in functionality to match the user's current level of expertise.

Another enabling technology on the horizon is tool integration facilities. For the most part, maintenance, testing, domain-specific modeling, project management and quality assurance tools are not well integrated with generic analysis, design and construction tools. Tool framework and repository definitions should be expanded to incorporate these functions in a seamless manner. Potential CASE users are looking for open environments spanning life cycle stages, development roles, distributed networks, and multivendor tools and computing platforms. Realizing the full power of second-generation tools will require data integration facilities (repositories) supporting high-volume management of small-grain objects with subsecond response in a distributed architecture. CASE repositories should support unlimited relationships among objects with full version and configuration management.

Lack of industry standards is inhibiting progress in CASE integration. Industry participants, especially computer systems manufacturers, could improve the situation by supporting full time, fully committed personnel on standards committees, by prototyping proposed standards before adoption, by adhering to standards in commercial products, and by agreeing on a common reference

model for future proposals and evaluations. Nevertheless, it is acknowledged that CASE integration standards are not mature and will continue to evolve in the future. Operational experience with a small number of alternative metamodels would bring us closer to realization of standard integration frameworks. CASE integration standards should be defined, in light of developments in related disciplines (such as electrical and mechanical engineering, electronic data interchange, multimedia, and imaging) to ensure appropriate interoperability, or at least the possibility of future extensions in those directions.

Anticipated challenges, beyond technical integration, include defining responsibility for maintenance and support in a multivendor environment. Also, a fully integrated CASE environment will result in a database of information that could be intellectually overwhelming. We will require effective strategies for complexity management to permit efficient control of scoping, viewability, navigation, data-sharing and security.

A third emergent area of technology is software reuse. Realizing the potential of software reuse requires more creative, comprehensive and sophisticated approaches than those employed to date. Effective software reuse depends on an overall reuse strategy that addresses software process standards, design techniques, tools, incentives, culture and economic models. The reuse concept is being extended beyond code modules to include object classes, programming clichés, design specifications, architectural blueprints, user interface templates, domain-specific rule bases, design histories, test suites, and so on. Reuse that spans projects, departments and even companies appears to be on the horizon. To encourage and enable reuse, we must find ways to record design alternatives and rationale, and to define more explicit specifications of form, fit and function. Reuse strategies must be supported by advanced reusable component management facilities at all conceptual levels.

Methods and Tools

We have a large arsenal of CASE tools to assist in many tasks associated with software development. It is certainly possible to increase the effectiveness of any development team with a judicious selection of point tools. Nevertheless, there is still room for considerable improvement. Our suite of tools needs to be expanded in breadth of coverage, particularly for requirements elicitation and design recovery. There is a shortage of support tools to aid in evaluating trade-offs associated with decisions to build or buy, maintain or reengineer, and reuse or start from scratch. We also need better tools for metrics collection and analysis, process modeling, and reuse management (e.g., library management for code, classes, templates and subsystems).

By and large, first-generation CASE tools, which deal mainly with the static aspects of software, have reached maturity. We are now looking forward to a more dynamic second generation of tools. Examples of these include collaborative requirements capture tools, simulators capable of executing system specifications, dynamic code analyzers, on-line design "advisors" and help systems, rapid prototyping environments, and interactive program understanding (i.e., reverse engineering) environments.

To continue the quest for defect-free software, we need to continue the pursuit of formal specification languages. By providing tools to transition between formal and semiformal representations, and to execute formal specifications, we can make formal techniques more acceptable and useful to the average software developer. More rigor will make possible the creation of tools for automatic test generation and code verification, and eventually, tools for software reliability and safety analysis.

Tools, and the environments in which they operate, should be scalable in terms of project complexity and size. They should be priced in a manner consistent with the enabled functionality, starting at a basic level and tracking the user's capacity to derive real value as his/her sophistication increases.

Trends and Priorities

The next IWCASE Workshop, to be held July 6–10 in Montreal, promises to bring greater clarity to these earlier findings. We will now offer a few personal observations about industry developments since the last workshop.

In the area of tool integration, Motif has become a *de facto* standard for CASE tool user interfaces in the Unix environment, while Presentation Manager and Windows 3.0 each seem to be holding their ground on PCs for the moment. With the adoption of PCTE in Europe as ECMA Standard 149, there has been some progress on data integration standards. Several groups in North America, including a PCTE User Group, NIST, PCIS and STARS, are looking at the feasibility of adoption in the U.S. Two European vendors, SFGL and Syseca, have announced Integrated Project Support Environments based on Emeraude's PCTE implementation.

Hewlett Packard's SoftBench has enjoyed good acceptance as a control integration facility in the U.S., and HP and IBM are working with several other vendors to further standardize it. A version of SoftBench integrated with PCTE is also expected. IBM has extended the definition of its AD/Cycle Information Model, and a few CASE vendors are beginning to actively use its facilities. DEC delivered its CDD/Repository, and a group of third-party tool vendors announced support for it. For the time being, the holy grail of an industry-wide tool integration standard continues to elude us, but there is more convergence in concept, and even implementation,

than is apparent from the suppliers' marketing strategies.

In the on-line business systems domain, the class of integrated toolsets that offer design-to-code capability remains small; most depend on the underlying capabilities of a database management system. Cross development is now supported by several of these integrated CASE products, freeing the developer from the target computer and allowing him/her to exploit the full capabilities and performance of engineering workstations (and high-end PCs), even for commercial development. Thus Unix- or OS/2-based tools will soon be able to target virtually any commercial computing system. The ability to generate client-server applications with multitargeted graphical user interfaces (Motif, Open Look, PM, Windows, Macintosh) is also being incorporated in these tools.

LAN-based team development is rapidly replacing timeshare configurations, although a central host may act as a corporate repository for shared design information. In general, the most advanced tools now provide more rigorous design rule checking and integrate their multiple design representations, or views, through a single cannonical form stored in a shared repository which may be centralized or distributed over a network.

Several commercial tools are now available that support metaCASE capability (i.e., the ability to completely redefine the method used, including visual representations, design rules, and target code generation optimization). These can adapt to vertical markets or unique in-house methods.

Reverse engineering tools now support database physical-to-logical derivations with analysis of live data, and interactive program understanding based on static and dynamic code analysis. Several tools support software "decomposition" (i.e., extracting all code from an application related to a specified function so it can be replaced, re-engineered or encapsulated and shared in a library).

Research in software quality processes and metrics continues, with announcement of some commercially available tools expected in 1992. Unfortunately, the state-of-the-practice in most organizations is still woefully lacking in this area. The related class of coordination technology, or "groupware," tools is also expanding with a growing number of network-based conferencing, planning, time management, and project management tools. In the future we expect effective tools for team engineering over wide geographical distances incorporating multimedia technology in the form of hypertext, graphics, images, animation, and verbal annotation.

Interest in reuse has increased, coincident with a growing realization that there may be no silver bullet, such as object orientation, that makes it easy. In general, effective reuse appears to require both extensive domain knowledge and a strategic commitment and plan on the part of the development organization. We are still exploring how design automation technology might assist in this process. However, large-scale reuse in the form of application templates seems to be emerging as a viable business, as companies begin to reuse and share their CASE design models.

Programming workbenches are becoming much more interactive and now provide more feedback on the dynamic behavior of code. New representations exploiting graphics and color are helping programmers in the traditionally difficult areas of complex dynamic data structures (e.g., array pointer dereferencing), proper memory allocation and deallocation, performance optimization, event-driven programming, multiprogramming, quality assurance testing, defect tracking, and configuration management.

Graphic user interface (GUI) tools are making it unnecessary for programmers to learn the intricacies of X-Windows, Motif, Open Look, PM or Windows. In many cases, the tools provide client-server database access and networking support, in addition to a direct manipulation facility for GUI creation and prototyping.

In the control systems area, CASE tools support systems-level prototyping and simulation. Some tools provide reusable components (encapsulated functions) accessed as objects in the form of iconic symbols. Systems can be built by assembling and connecting components and automatically generating executable code.

With the introduction of so much new technology, the biggest problems with CASE continue to be the fragmented nature of the tools and methods, and the need for more mature organizational frameworks in which to apply it. CASE '92 will have a strong focus in these areas in an effort to discover ways to remove the barriers and exploit the possibilities of CASE more effectively.

About the Guest Editors:

GENE FORTE is president of CASE Consulting Group, and executive editor of the CASE OUTLOOK® international report on computer-aided systems engineering. As a researcher and analyst, Forte specializes in the latest tools and techniques for commercial and real-time systems planning, development, management and quality control. He is currently program cochair for CASE '92. **Author's Present Address:** CASE Consulting Group, 11830 Kerr Parkway, Suite 315, Lake Oswego, OR 97035, g.forte@compmail.com.

RONALD J. NORMAN is an associate professor of information and decision systems at San Diego State University, where he teaches courses in information system management, systems analysis and design, and information as an organizational resource. Norman served as general chair for CASE '90. His research interests include CASE technology, technology transfer and organizational change issues, and object-oriented systems analysis and design. **Author's Present Address:** College of Business Administration, San Diego State University, San Diego, CA 92182-0127, rnorman@sciences.sdsu.edu.