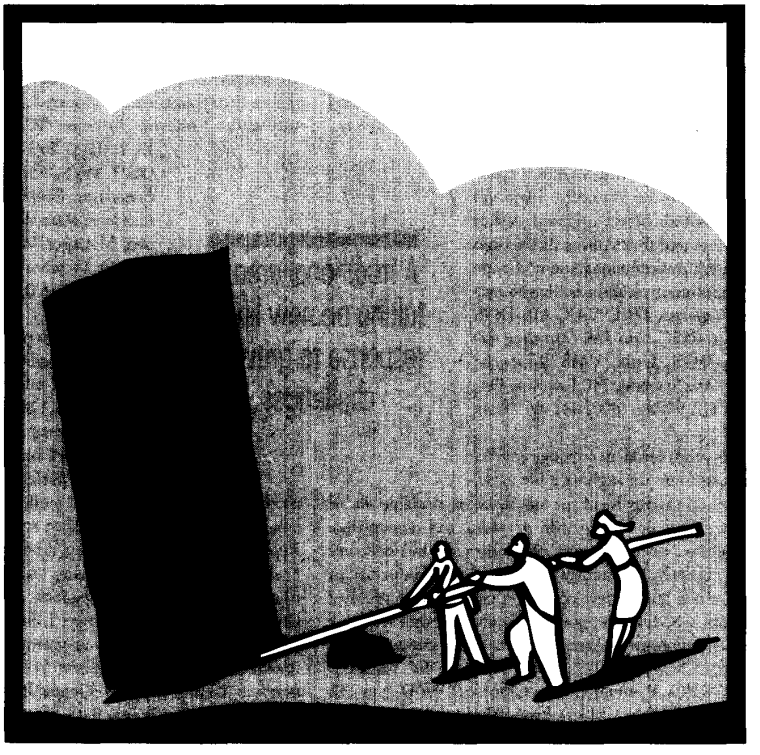
FEATURE

AN INTEGRATED ENVIRONMENT FOR REQUIREMENTS ENGINEERING

Studies show that the greatest leverage to improve quality lies in supporting the collection of correct, unambiguous requirements. Such an environment must support collaborative work.

JAMES D. PALMER and N. ANN FIELDS
George Mason University



To improve quality, software developers need comprehensive techniques that use systematic, integrated, consistent, and enterprise-wide approaches. Productivity, quality, and sound project management are functions of how well an integrated support environment encourages the best use of people, policies, processes, and plans.

If we are to deliver high-quality software that meets the customers' needs, we must apply integrated support for development at the point of greatest leverage. Recent studies indicate that the greatest leverage points to influence the production of quality software are coincident with the phases in which we expend the fewest resources.[1]

These studies show that five percent of the total cost of a major system is expended on design and development, yet 70 percent of the ability to influence quality comes from this meager amount. In terms of effort, requirements analysis consumes merely five percent, but affords 50 percent of the leverage to influence improved quality. In terms of preproduction costs, system definition absorbs five percent, yet again provides 50 percent of the leverage to influence quality.

In other words, it takes 100 times more effort to correct errors in requirements

0740-7459/92/0500/0080/$03.00 © IEEE

discovered at the time of coding than it does to correct the same errors if they are discovered during requirements engineering.[2]

Developers have long given lip service to the development of high-quality requirements, yet this area is rarely a focus of research. We believe the value added in producing high-quality (correct, testable, and unambiguous) requirements is sufficiently great to warrant concentration in this area.

OUR ENVIRONMENT

We have developed an integrated environment for requirements engineering that supports participatory development activities. Our environment helps ensure quality by supporting and encouraging group participation and interaction.

Our computer-supported cooperative-work environment supports the development and analysis of system- and software-level requirements for large, complex applications. It encompasses and coordinates all aspects of requirements development, from conceptual inspiration, through planning, to specific project details.

Case studies of groups using this environment show it can support requirements engineering for groups of users determining system-level requirements for large, complex systems. It can also support the interactive and iterative activities that take place between users and requirements-engineering teams, including

- requirements elicitation,
- classification,
- analysis,
- traceability,
- validation, and
- design

which can lead to test-plan generation. Its open architecture means you can integrate commercial CASE tools as appropriate, and, although it was designed to support requirements development, there's no reason you can't extend it to other phases of the development life cycle.

SUPPORTING GROUPS

Most large, complex systems-engineering projects are inherently group efforts undertaken by developers, managers, and users who represent development and application from all perspectives.

Requirements engineering is the elicitation, analysis, classification, and design of systems and software requirements. It incorporates many functions, methods, and approaches applied by users and designers.

In an evolutionary life cycle, requirements engineering is a major, continuous activity whose most important function is to specify a system that will meet the users' perceived needs. We have found it is desirable and necessary to involve users in requirements elicitation and validation to ensure that their view of the system is represented correctly.

Our environment has two capabilities that are key to the elicitation of correct, complete, and unambiguous information:

- it can help you obtain useful information from individuals or groups and
- the information it captures can be represented in appropriate media formats.

To help the requirements engineer obtain useful information from users, we've incorporated small-group interaction techniques in the environment and developed tools to help support group interaction. To let engineers represent the captured information in appropriate formats, our multimedia environment supports the capture and manipulation of information as text, graphics, audio, and video.

Our multimedia-based environment uses rapid prototyping and takes into account the *content*, not just the form, of requirements. It provides for the presentation of concepts and prototypes that lead to formal specifications. Its CASE tools let users manage problems like imprecise, ambiguous, and incomplete requirements.

ARCHITECTURE

Our environment runs on an Apple Macintosh FX with 8M bytes of RAM, 160 Mbytes of disk storage, analog video I/O, audio I/O, a still digital camera, and other peripherals. We developed software that uses object-oriented programming environments, including SuperCard, HyperCard, and Smalltalk. We designed, developed, and applied various CASE tools to deal with requirements that are imprecise, ambiguous, or conflicting.

The environment supports the primary activities of requirements classification, indexing, and clustering for analysis; validation; test-plan generation; quality assurance; and traceability of system-level requirements throughout the life cycle.

Figure 1 shows the environment's architecture, which supports multiuser, multidesigner dialogue that includes the traditional textual and graphical information sources, plus nonstandard sources like audio, video, and animation. Figure 2 shows the information flow within it.

The environment's essential components are the CASE tools we designed: The COSP object manager; the Lexscan lexical scanner and analyzer; the Knowledge-Based Requirements System knowledge-based conflict-resolution tool; and the Costtool cost-estimation tool.

COSP. The COSP object manager *captures*, *organizes*, *synthesizes*, and *presents* information.[3] The capture function involves all elicitation processes, but emphasizes interactive user-designer groups. You can use the environment to capture informa-

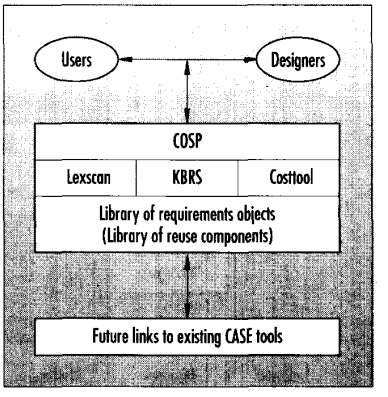


Figure 1. Architecture for the requirements-engineering environment. COSP lets the user manage objects in the system. The environment includes three CASE tools: the Lexscan lexical scanner and analyzer; the Knowledge-Based Requirements System conflict-resolution tool; and the Costtool cost-estimation tool.

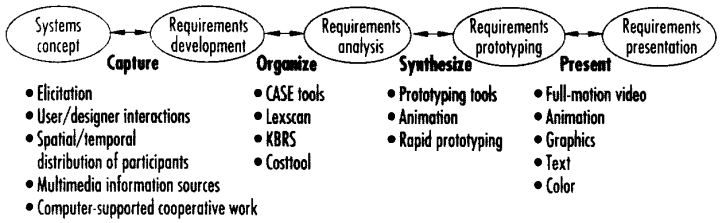


Figure 2. A simplified diagram of the information flow in our environment. Once requirements statements are analyzed and as many flaws have been corrected (or noted for correction later) as possible, rapid prototyping begins. These prototypes are then presented to the users and developers in whatever form they desire.

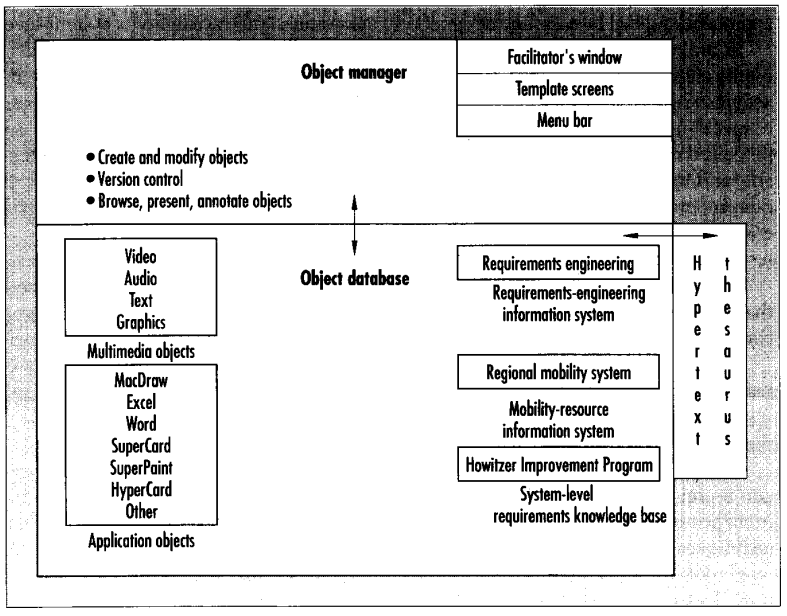


Figure 3. The object-management system manages all the information in the environment. In the object manager, you create and modify objects, create and maintain version control, browse objects, present objects dynamically, and annotate them. The object database contains all objects, applications, and domain-specific knowledge and data.

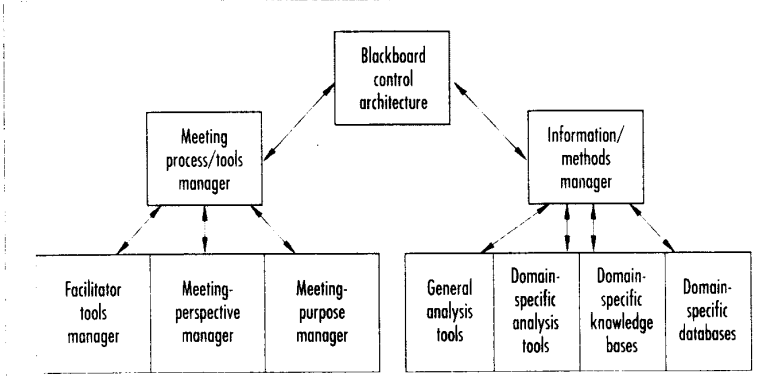


Figure 4. The computer-supported, cooperative-work environment is managed through a blackboard-control architecture and includes two major subsystems, one to control meeting processes and tools and another to control information and methods.

tion from groups that are spatially and temporally distributed and from all types of media, through associative dynamic links that relate any object set with any other object set. Our latest work has focused on expanding this capture component to support collaborative work, as we detail later.

The organize function encourages dynamic associative links with objects that support a particular requirements function. The synthesize function lets you prototype requirements at any time, and the present function supports many exhibition formats that you can tailor to users and designers.

All information in the environment is converted to objects and managed by an object-management system, shown in Figure 3. In the figure, the object manager contains the facilitator's window, template screens for producing additional objects, and a menu bar for navigation. With the manager, you can create and modify objects, create and maintain version control, browse objects in any order, present objects dynamically, and annotate them.

The object database contains all multimedia objects, various applications, and domain-specific knowledge bases and databases (such as the requirements-engineering information system, a mobility-resource-information system, and a system-level requirements knowledge base).

Lexscan. This tool analyzes the syntax of natural-language statements.[4] It automatically classifies requirements by applying indexing and clustering techniques. Its major objective is to aggregate a set of N requirements into a set of M requirements clusters such that M is significantly less than N.

Lexscan uses a two-tiered clustering algorithm to discriminate among a set of M requirements statements and group them according to their similarities. The two-tiered clustering algorithm provides the information necessary to successfully differentiate between similar and dissimilar requirements and cluster similar requirements. It appears to be much more flexible than other indexing schemes, such as those based on facets and predefined taxonomies.

KBRS. Once requirements are classified, the Knowledge-Based Requirements System analyzes them to determine conflicts, incompleteness, inconsistencies, and imprecision within and across requirements clusters.[5]

KBRS is a knowledge-based CASE tool designed to examine the use of imprecise and ambiguous words, detect conflicts among quality-metrics terms, and present these problems to the user for clarification and resolution. Once these problems are clarified or resolved, the requirements engineer can tag each statement for traceability and assign metric primitives with the help of another CASE tool. The engineer also assigns test tools and develops a test plan to validate requirements statements using these metrics primitives.

Costtool. This effort-estimation tool uses the measure of a development group's average productivity modified by such influences as the introduction of a new CASE tool, the group's experience level, and their familiarity with the project.[6] Developed for object-oriented design, Costtool uses classes and methods counts as input.

The general form of Costtool's effort-estimation equation is

A_X = [relative effort for the part of the project not affected by X]
× [portion of the project not affected by X]
+ [relative effort for the reapplicable part of X]
× [portion of the project involving X that can be reapplied to other projects]
+ [relative effort for the project-specific part of X]
× [portion of the project involving X that is project-specific]

Object library. The output of these analyses is a set of validated requirements statements. The engineer places these statements into a library. Because our environment uses classification and clustering algorithms, it can describe requirements specifications and store them according to function and keyword so they can be retrieved by those keys. Thus, our environment supports the reuse of requirements specifications.

Prototyping. Once the requirements statements have been analyzed and as many flaws as possible have been corrected (or noted for correction later), rapid prototyping of the system developed to date begins. Prototyping tools include various structured design tools, animation tools, and screen-development tools. These prototypes are then presented to the users and developers in whatever form they desire.

COMPUTER-SUPPORTED COOPERATIVE WORK

An integrated environment designed for computer-supported cooperative work must support interactive information-resource development, information analysis and retrieval, and techniques that support group decision-making, including conflict resolution and consensus building.

Figure 4 shows the conceptual architecture of the computer-supported, cooperative-work environment.[7] The system is managed through a blackboard-control architecture and includes two major subsystems, one to control meeting processes and tools and another to control information and methods.

Figure 5 shows the meeting manager, which supports the meeting facilitator and general meeting conduct. Through the meeting manager, the facilitator commands several configurations to control the presentation, including control of the network configuration, public screen, agenda, and evaluation method. The meeting manager also contains tools to help capture a meeting's context.

The other parts of the meeting-manager subsystem keep records so subsequent users can understand how decisions were reached and what the meeting's dynamics were. For example, if the meeting was billed as a decision-making meeting to be conducted in a rational-actor mode, but turned out to be an information meeting with an organizational perspective, this would be recorded. Later, you could analyze the meeting's outcome to determine if the group's will was recognized or compromised or if the change in perspective affected the outcome.

Figure 6 shows the decision-meeting manager, which is a component of the meeting-manager subsystem. In this component, we track the type of meeting, the decision strategy used, and the users' inquiry system.[8] This information is re-

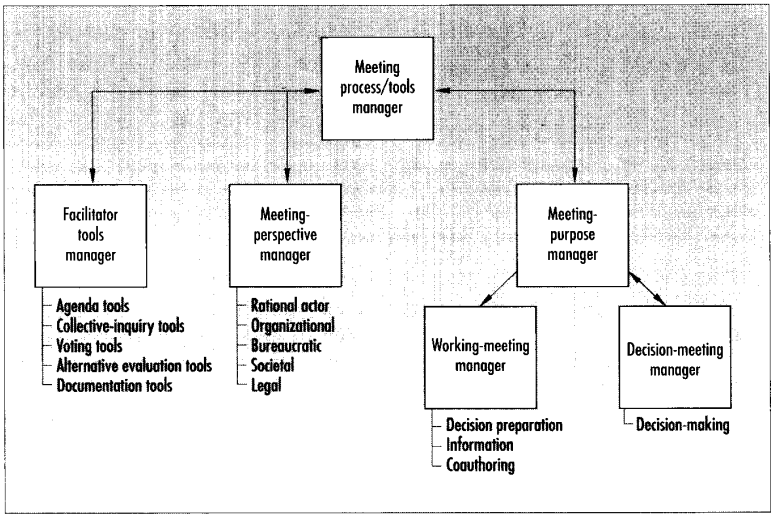


Figure 5. The meeting-manager subsystem supports meetings. The facilitator's tools are designed to control the presentation, network configuration, public screen, agenda, and evaluation method and help capture a meeting's context. The perspective manager and purpose manager record information about decisions and how they were reached so that later users can understand the meeting's dynamics.

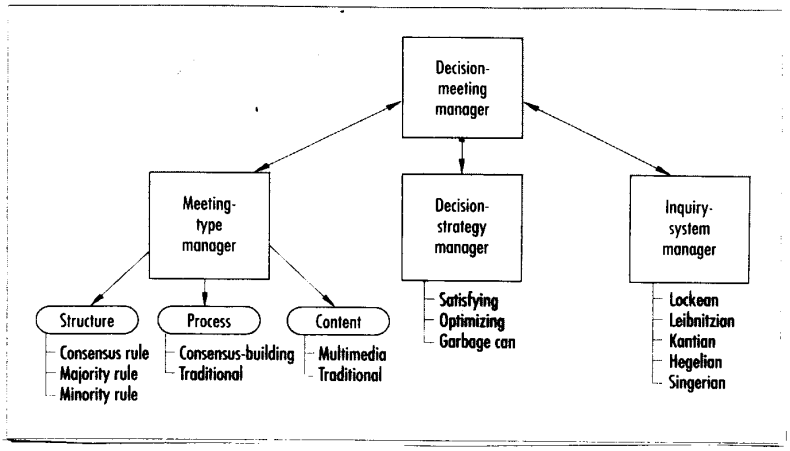


Figure 6. The decision-meeting manager tracks the type of meeting, the decision strategy used, and the users' inquiry system.

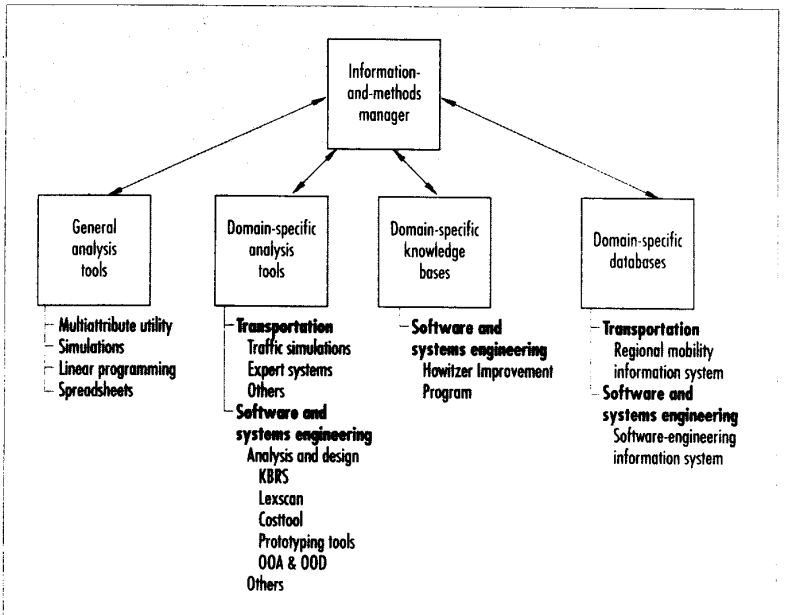


Figure 7. The information-and-methods manager provides general and domain-specific analysis tools. It currently includes domain-specific tools for transportation systems and system requirements. The current domain-specific knowledge base is the US Army's Howitzer Improvement Program; the domain-specific databases are requirements engineering and regional mobility.

turned to the meeting process and tools manager.

Figure 7 shows the information-and-methods manager. This subsystem provides several general analysis tools, including multiattribute utility, and domain-specific analysis tools. Currently, it includes tools for two domains (transportation systems and system requirements); it can be configured to include other domain-analysis tools as needed. For example, when the computer-supported cooperative-work environment is used to help generate system requirements, the information-and-methods manager can access Lexscan, KBRS, and Costtool to help the user understand the domain problem and generate a correct, unambiguous requirements specification.

In Figure 7, the current domain-specific knowledge base is the US Army's Howitzer Improvement Program; the domain-specific databases are requirements engineering and regional mobility.

Experiments. We've experimented with the computer-supported cooperative work environment with five decision-making groups, each consisting of four to nine participants, for a total of 32 participants.[7]

Three groups comprised a total of 19 PhD students and three faculty members. Each group met for three hours to generate a prioritized list of research objectives for requirements-engineering research for the next two to five years at George Mason University.

The fourth group comprised six senior executives who met for three hours to develop the requirements for a study on the future of nuclear power in the United States. This group set its own agenda — it was not assigned a decision problem, as the other groups were.

The fifth group comprised four master's students who met for two hours to generate a prioritized list of solutions to the campus parking problem, similar to the problem detailed by C.F. Gettys and colleagues.[9]

Evaluation. To collect data on the effectiveness of the decision-support environment, we videotaped the meetings and had the participants complete postsession questionnaires.

The questionnaire asked participants to indicate their agreement with a series of statements using a Likert scale, in which 1 means "strongly disagree" and 5 means "strongly agree." The statements were designed to assess the decision-making process, satisfaction with previous meeting experiences, satisfaction with the current meeting experience, and satisfaction with and appropriateness of the computer-aided tools used. All 32 participants completed the questionnaire.

The first section of the questionnaire, modified from an earlier experiment,[10] included 12 statements about the value of computer-aided decision-making in helping the group generate ideas and achieve a goal, the participants' commitment to and confidence in the decision reached, and their satisfaction with the process and the outcome. In answering this section, 56 to 90 percent of the participants rated the

decision-support environment favorably — indicating "agree" or "strongly agree" to statements like "The computer-aided decision-making process helps the group achieve its goals."

The statements in the next section had the participants evaluate the specific computer-aided tools used. In response to these four statements, 81 to 97 percent of the participants rated the tools favorably — indicating "agree" or "strongly agree" to the statement that the specific tool was of value.

The statements in the next section dealt with the value of meetings in helping participants understand the problem, understand why decisions are made, what the decisions are, and whether meetings let them express their concerns. This section had three sets of statements in which participants were asked to rate their experience in previous meetings (five statements) and in the current meeting (five statements) and how the previous and current meetings compared (four statements). In response to these statements, 63 to 91 percent of participants rated the computer-aided environment favorably, while 35 to 58 percent rated previous meetings favorably.

The small number of participants in this initial evaluation limits the interpretation of the findings, but these results do indicate that decision-makers find the decision-making process and the computer-aided tools used in this environment both helpful and satisfying.

Our environment can accommodate very large amounts of highly complex data from a variety of media sources. We have included video, audio, text, and graphics into a single workstation and have provided for the capture, organization, synthesis, and presentation of this varied material. Our system takes advantage of a model-based management system to run simulation and analytical models and to combine results for presentation.

Sessions with groups working on very different requirements problems show that the decision-support environment is a powerful adjunct to the facilitator in building a consensus. The fact that each member of a group can deal with the same information and has equal ability to have the information organized, reorganized, synthesized, and resynthesized on demand facilitates consensus building.

Clearly, the decision-support environment is neither a substitute for human facilitators and decision-makers, nor any better than the quality and quantity of relevant information it is fed. However, our experience with it indicates that it can enhance and extend the basic tools a facilitator uses to guide a group to a consensus.

Our tests on the computer-supported cooperative work environment have led us to believe that computers can help improve decision-making and help meeting facilitators build a consensus. In the near future, we plan to develop more robust requirements-engineering scenarios and test the computer-supported cooperative work environment more extensively to determine its efficacy in assisting software developers. ♦

ACKNOWLEDGMENTS

The research and development of this environment was sponsored in part by a grant from the Virginia Center for Innovative Technology.

REFERENCES

1. A.S. Shumskas, "Software - TQM, T&E, OSD, and You," *Proc. Nat'l Symp TQM for Software*, Nat'l Inst. for Software Quality and Productivity, Washington, DC, 1991.
2. B.W. Boehm, "Improving Software Productivity," *Computer*, Sept. 1987, pp. 43-57.
3. P.H. Aiken, *A Hypermedia Workstation for Requirements Engineering*, doctoral dissertation, George Mason University, Fairfax, Va., 1989.
4. J.D. Palmer, Y. Liang, and L. Wang, "Classification as an Approach to Requirements Analysis," *Advances in Classification Research and Application: Proc. 1st ASIS SIG/CR Classification Research Workshop*, S.M. Humphrey and B.H. Kwasnik, eds., Learned Information, Medford, N.J., 1990.
5. D. Samson, *Automated Assistance for Software Requirements Definition*, doctoral dissertation, George Mason University, Fairfax, Va., 1988.
6. S.L. Pfleeger, *An Investigation of Cost and Productivity for Object-Oriented Development*, doctoral dissertation, George Mason University, Fairfax, Va., 1989.
7. N.A. Fields, *An Evolutionary Group Decision Model for Computer Supported Cooperative Work*, doctoral dissertation, George Mason University, Fairfax, Va., 1991.
8. C. Churchman, *The Design of Inquiring Systems: Basic Concepts of Systems and Organizations*, Basic Books, New York, 1971.
9. C.F. Gettys et al., "An Evaluation of Human Act Generation Performance," *Organizational Behavior and Human Decision Processes*, Feb. 1987, pp. 23-51.
10. A.R. Dennis et al., "Bringing Automated Support to Large Groups: The Burr-Brown Experience," *Information & Management*, Mar. 1990, pp. 111-121.

James D. Palmer is the BDM International professor of information technology at George Mason University. His research interests are software engineering, group-decision support systems, multimedia systems, and requirements engineering. He has written three books and more than 75 papers.

Palmer received a BS and an MS in electrical engineering from the University of California at Berkeley and a PhD in electrical engineering from the University of Oklahoma. He is a fellow of the IEEE and its Systems, Man, and Cybernetics Society and the IEEE Computer Society.

N. Ann Fields is a research assistant professor of systems engineering in the School of Information Technology and Engineering at George Mason University. Her research interests are group-decision-support systems, multimedia systems, and requirements engineering.

Fields received a BS in mathematics from the University of Miami, an MS in operations research and statistics from the University of Southern Mississippi, and a PhD in information technology from George Mason University.

Address questions about this article to Palmer at the Center for Software Systems Engineering, School of Information Technology and Engineering, George Mason University, Fairfax, VA 22030-4444; Internet jpalmer@gmuvax.edu.