Requirements are precise statements of need intended to convey understanding about a desired result. They describe the external characteristics, or user-visible behavior, of the result, as well as constraints such as performance, reliability, safety, and cost. A collection of related requirements represents both a model of what is needed and a statement of the problem to be solved.

Requirements engineering is a systematic approach to the development of requirements through an iterative process of analyzing the problem, documenting the resulting requirement insights, and checking the accuracy of the understanding so gained. A requirements engineering environment must provide the requirements engineer with appropriate mechanisms to facilitate the analysis, documentation, and checking activities. What this means and how it is accomplished, now and in the future, is the subject of this special issue.

Analyzing the problem

Analysis is the systematic process of reasoning about a problem and its constituent parts to understand what is needed or what must be done. Analysis also involves communicating with many people. Initially those who are most familiar with the existing need and its surroundings, that is, the problem domain must be contacted. Requirements engineers must also communicate with those who are working...
to satisfy the need and with those who will pay for them. They will also need to communicate with the users, managers, and maintainers because they are all potential sources of new requirements.

The requirements engineering environment must support analysis in several ways. First, the reasoning process must be guided by an analysis methodology appropriate to the problem. The environment should enforce the procedures of the methodology and facilitate its application through an appropriate supporting work place. The work place must provide the requirements engineer with the software tools needed to gather the information necessary to reason about and understand the problem domain. Such tools must include rigorous, but natural ways to describe models of real-world problem domains. Not only must they store and maintain consistent model descriptions, but they must also capture basic information concerning the problem itself to assist the requirements engineer in asking and resolving questions about model behaviors.

Second, information must be organized to permit quick locating and easy access to pertinent facts. An environment that facilitates locating and accessing is necessary for stimulating the insights which produce requirements.

Third, communications between the requirements engineer and those in the community of interest will take place at scheduled, face-to-face meetings, as well as at spontaneous meetings or in phone conversations as the need arises. The requirements engineer’s environment must support the presentation and discussion activities by providing the communications network, the work place to facilitate its easy and rapid access, and the audio/visual tools—all of which are needed for successful interchange.

Documenting the requirements

While analyzing user needs, the requirements engineer documents several different types of information because the requirements must capture and convey the overall scope of a problem, the semantics of its important objects and activities, their relationships and their connections with the problem domain. Since the documentation is needed by the requirements engineer and others involved in the formulation and solution of the problem, it must be presented from different perspectives and with different audiences in mind.

Completeness checking requires examination of the problem as perceived by those who are most familiar with it, so that the result will satisfy the real need. It demands skillful interchange between the requirements engineer and those who best understand the problem but are probably least familiar with the formal representation of the problem statement.

The requirements engineer’s environment must support the checking process by providing tools for understanding and communicating. The end result will be a refined understanding of the problem.

The requirements engineer’s environment must support the checking process by providing tools for understanding and communicating. Much of the checking process is labor-intensive and can be assisted by graphically presenting the requirements to enhance understanding and by simulating requirements to predict the consequences of their implementation. In addition, the environment must retain data from the problem domain to permit responses to questions concerning the essential objects and activities, how they are related, and why. The end result of the checking process will be a refined understanding of the problem, an understanding that will be the basis for subsequent analysis and eventually a completed statement of the problem.

In this issue

The requirements engineering environment outlined above is more a concept than an operational capability. However, excellent work has been done in developing requirements analysis models, languages, and tools, and these efforts contribute to the environment.

We begin with “A Taxonomy of Current Issues in Requirements Engineering” by Gruia-Catalin Roman of Washington University in St. Louis. Roman presents developments in requirements methods and techniques within a framework for understanding the essential components of a require-
environments methodology. This framework also serves as a departure point for an examination of the outstanding issues of current requirements technology—those fundamental problems that must be resolved before the requirements engineering environment can be considered an engineering problem.

The next four papers provide an in-depth examination of three requirements engineering techniques originally developed during the mid-1970's, which have since evolved toward comprehensive requirement engineering environments. The first is the structured analysis and design technique presented here by its principal architect Douglas T. Ross, who introduced it in 1976, after a three-year development and application period. Ross opens his paper, "Applications and Extensions of SADT," with a tutorial on this method for documenting the architecture of large and complex systems. He describes the discipline and mechanisms of SADT, how it is practiced, why it works, and what its practical limits are. He summarizes experiences with SADT on a number of diverse projects and concludes with descriptions of several instances in which SADT has been combined with other techniques and tools to provide more comprehensive system development environments for specific projects.

Next, Mack Alford presents the third in a series of reports on the maturing process which the Software Requirements Engineering Methodology, or SREM, has undergone since its 1976 introduction. Entitled "SREM at the Age of Eight: The Distributed Computing Design System," this paper describes how SREM has evolved into a software development life-cycle environment for defining and implementing distributed systems. Not only does it describe a system requirements analysis methodology and how it interfaces to the SREM, but Alford's paper explains for the first time how the products of these methodologies evolve into design, implementation, and testing activities.

While Alford's paper presents SREM from the perspective of an environment developer, the following article, "A Case Study of SREM," presents a user's view. Paul Scheffer, Albert Stone, and William Rzepka describe their experiences in learning and using one of the few software requirement engineering environments currently available. Their evaluation includes both a qualitative assessment of the strengths and weaknesses of SREM and some interesting quantitative results that serve to characterize the requirements engineering process in this kind of environment.

This issue explores work responding to the needs of the requirements engineering environment: the foundation for modeling user needs has been laid.

The next article reports on the evolution of the Input/Output Requirements Language (IORL) developed in the early 1970's as a system engineering specification technique. Its simple graphical/tabular approach for recording requirements has been retained as the notation for "Specification-Based Software Engineering with TAGS," as reported here by Gene Sievert and Terrence Mizell. The authors describe IORL, its role in a system-development methodology based on prototyping, and some experiences in developing requirements documentation with IORL.

Stephanie White and Jonah Lavi report on the findings and conclusions of an "Embedded Computer System Requirements Workshop" in the following article. Held last fall at the Naval Research Laboratory, the workshop attracted requirements engineers, software development managers, and computer scientists who have been developing and applying various formal requirements models to the analysis of requirements for embedded computer systems. Participants assessed the progress in model development and the effectiveness of the models and arrived at a general agreement on some issues, disagreed on others, and identified several issues needing additional thought. Among other issues, they agreed on the use of finite-state machine representations to model embedded computer systems and the need to model both the system and its environment, ideas that are central themes of our final two papers.

The first of these, by M. Chandrasekharan, B. Dasarathy and Zen Kishimoto, describes an environment for specifying and validating the functional and performance requirements of telecommunications-type, real-time systems. "Requirements-Based Testing of Real-Time Systems: Modeling for Testability" shows how a finite-state machine model is used as the basis for a behavioral description of telecommunications systems and how that description generates test scenarios which are executed as part of the system validation process.

Current requirement languages make possible a formal statement of system requirements. With a view toward the future, our final paper, "Knowledge Representation as the Basis for Requirements," recognizes that system requirement statements alone are insufficient for understanding what users need. Observing that the system built to satisfy the need exists in and affects the users' world, Alexander Borgida, Sol Greenspan, and John Mylopoulos argue that a portion of the world in which the problem exists must also be captured in the requirements specification. The authors then describe how requirements languages must represent world knowledge in a natural and convenient form, so that it is understandable by both the system developer and the user. Their approach exploits concepts that have evolved from research in artificial intelligence and software engineering to provide new directions in requirements engineering.

While the above work contributes much to the environment's foundation, much remains to be done, especially in the areas of modeling the problem domain and insuring that results meet expectations. It has been our
purpose in this special issue to explore work already responding to the environment's needs, as well as to explore promising new ideas. In doing so, we hope to stimulate new work that will move a complete environment closer to reality.

Acknowledgments

The guest editors would like to thank the participants of the 1982 Kyoto Symposium on Current Issues of Requirements Engineering Environments who provided the stimulus for this issue, the authors who submitted articles, and the referees whose comments and suggestions were extremely helpful. Special thanks go to C. V. Ramamoorthy, Catalin Roman, Bill Riddle, Ray Yeh, Roc Luorno, and Debbie Cerino for their valuable comments, to Sam DiNitto for his support throughout this undertaking, and to Grace Martin for her help in organizing it.

William E. Rzepka has been employed since 1967 at the Rome Air Development Center, where he is a computer scientist in the Command and Control Division. In his current assignment in the Software Engineering Section, he is responsible for the evaluation and development of system and software definition technologies. His technical interests include requirements analysis and design methodologies, prototyping, and system definition workstation technology. He previously worked in the areas of operating systems, database management systems, and computer security. Rzepka earned his BS in electrical engineering and MS in systems and information science at Syracuse University in 1964 and 1972, respectively. He is a member of ACM and IEEE Computer Society.

Questions concerning this special issue can be directed to Rzepka at RADC/COEE, Griffiss Air Force Base, New York 13441-5700.

Yutaka Ohno has been a professor of information science at Kyoto University since 1972. His research interests include software design methodology, performance evaluation, simulation, and language design for system description. He was formerly a member of the research staff of the Railway Technical Institute of the Japanese National Railway. From 1976 to 1978, he was a vice president of Information Processing Society of Japan. He was co-chairman of the Third USA-JAPAN Computer Conference and General Chairman of the Sixth International Conference on Software Engineering. He is currently President of the Japan Society for Software Science and Technology. He received prizes from the IEEJ in 1960, IEEEJ in 1968, and the Japanese Government in 1971. Ohno received his BE in mechanical engineering and PhD in electronic engineering from the University of Tokyo in 1946 and 1962, respectively.

OPPORTUNITIES FOR RESEARCH IN SIGNAL AND IMAGE PROCESSING

Honeywell is searching for a few select senior-level and entry-level professionals to conduct advanced research and development in Signal and Image Processing at its Systems and Research Center, located in the beautiful metropolitan area of Minneapolis/St. Paul, Minnesota. The Systems and Research Center is the high-technology R&D organization within Honeywell Aerospace and Defense. We provide an opportunity to do futuristic research and development in an unregimented, university-like atmosphere.

Candidates should have a Master's or Doctorate degree in Electrical Engineering, Computer Science, or related area. Background should include:

- Signal and Image Processing for MMW Radar, SAR, IR, Laser and Acoustic Sensors
- Target Identification and Tracking Algorithms
- Artificial Intelligence for Fusing Information from Radar Sensors, FLIR, Laser and Acoustic Sensors
- Speech Processing
- Advanced Architectures for Real-Time Implementation of Signal and Image Processing Algorithms

We provide competitive salaries for candidates with appropriate experience, an excellent benefits package, a liberal education policy with the opportunity to interact with a top-notch university, and rapid career advancement for high achievers. U.S. Citizenship required; will consider resident aliens who are within 6 months of obtaining U.S. Citizenship. Please call or send resume to:

Dr. Raja Suresh, Section Chief
Multisensor Signal Processing
Honeywell Systems and Research Center
P.O. Box 312 MN17-2361-RSI
Minneapolis, MN 55440
Tel: 612/378-4579

Together, we can find the answers.

Honeywell
An Equal Opportunity Employer M/F/H/V