GUEST EDITORS' INTRODUCTION

Giving VOICE to Requirements Engineering

♦ Developers have plenty of reasons to avoid investing in requirements engineering: It is next to impossible to capture user needs completely, and needs are constantly evolving. But can we afford to shut out the customer?

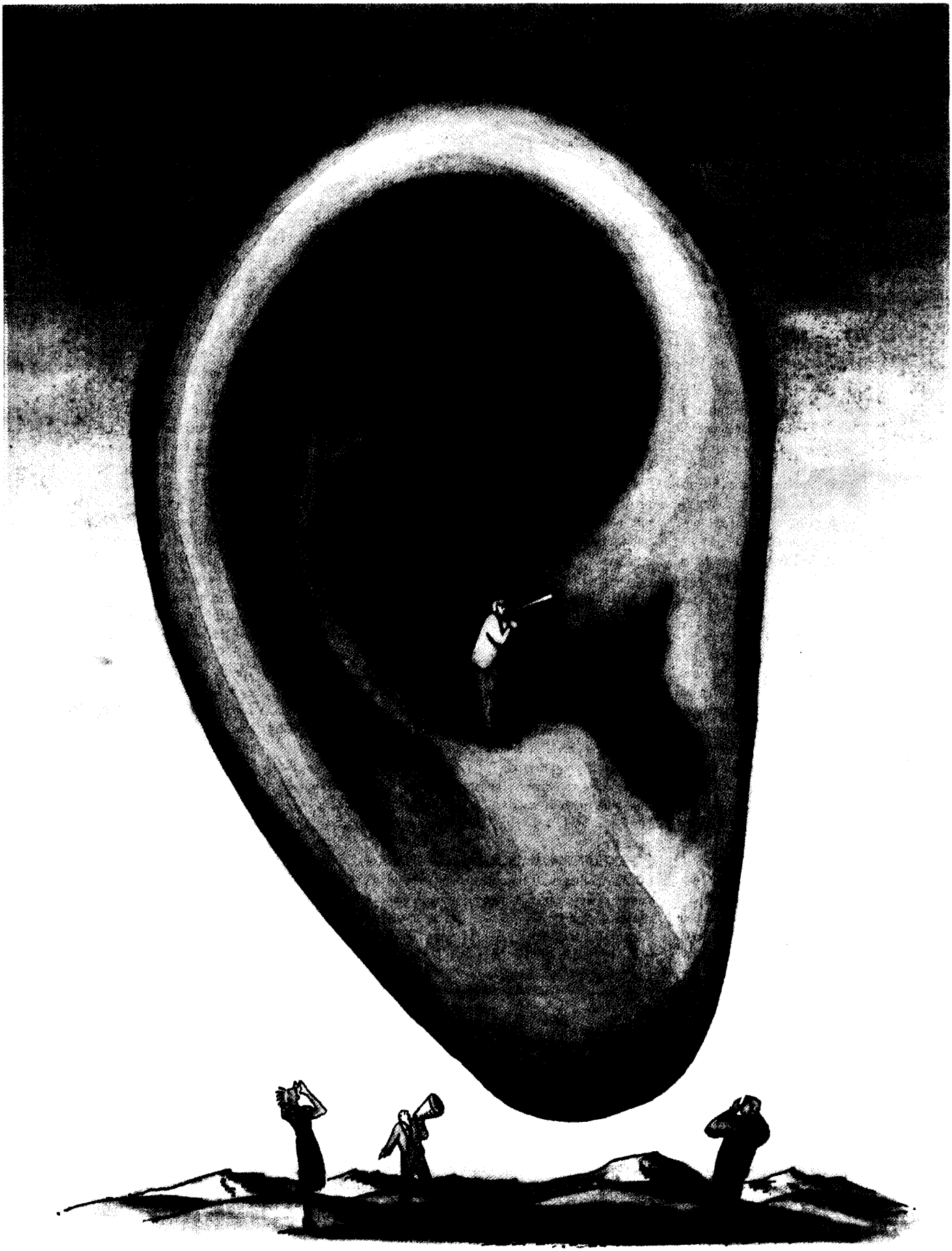
ALAN M. DAVIS, *University of Colorado at Colorado Springs*
PEI HSIA, *University of Texas at Arlington*

The gap between software research and practice is no more evident than in the field of requirements engineering. In theory, requirements engineering applies proven principles, techniques, languages, and tools to help analysts understand a problem or describe a potential product's external behavior. In practice, many software customers understandably wonder if anyone is listening.

The technology-transfer gap is a factor in every software-engineering field, but the problem with requirements engineering is more insidious. Even if the research community could formulate the "best" solutions to industry's problems, they most likely could not overcome the biggest problems in requirements engineering today:

♦ Needs are often impossible to realize until after a system is built.

0740-7459/94/$04.00 © 1994 IEEE



SIGNPOSTS AND LANDMARKS: A REQUIREMENTS ENGINEERING READING LIST

To learn more about requirements engineering, I suggest you start with some overview books and branch out to periodicals, proceedings, and some classic papers.

OVERVIEW BOOKS. Four books take a broad view of the principles of requirements engineering, instead of pitching a particular approach.

For good insights on how to determine what problem you are really trying to solve, read Donald Gause and Gerald Weinberg's *Are Your Lights On?* (Dorset House, 1990). Weinberg contributes more of his valuable insights into the role of performing both requirements and design in *Rethinking Analysis and Design* (Dorset House, 1988).

Gause and Weinberg teamed up again to produce *Exploring Requirements* (Dorset House, 1989), an excellent survey of elicitation and problem-discovery techniques. And my own book, *Software Requirements: Objects, Functions and States*, (Prentice-Hall, 1993) surveys and compares many analysis and specification techniques.

There are many dozens of other books on the market about specific approaches, be they object-oriented, state-oriented, or structured.

PERIODICALS. This special issue represents the fifth issue the IEEE Computer Society has devoted to requirements engineering:

The January 1975 issue of *IEEE Transactions on Software Engineering* includes some of the great "early" work on requirements engineering, including landmark papers on SADT, REVS, and PSL/PSA.

In May 1982, *Computer* captured the next generation of requirements-engineering technology, primarily tools and languages.

By April 1985, some methods papers (SADT, SREM, and TAGS) were included among the languages and testing articles in a special issue of *Computer*.

Finally, the March 1991 issue of *IEEE Transactions on Software Engineering* included reports on two relatively mature research efforts, Paisley and RA, and one new research effort to precisely define what completeness is in finite-state, machine-based specifications.

PROCEEDINGS AND CLASSIC WORKS. Another excellent source of information are proceedings, including *Specifications of Reliable Software* (IEEE CS Press 1979), *Proc. Int'l Symp. on Current Issues of Requirements-Engineering Environments* (North-Holland, 1982), *Proc. IEEE Int'l Symp. on Requirements Eng.* (IEEE CS Press, 1992), and *IEEE Int'l Conf. on Requirements Eng.* (IEEE CS Press, 1994).

Finally, some papers are classics in the requirements-engineering field:

♦ V. Basili and D. Weiss, "Evaluation of a Software Requirements Document by Analysis of Change Data," *Proc. Fifth Int'l Conf. of Software Eng.*, IEEE CS Press, 1981, pp. 314-323.

♦ T. Bell and T. Thayer, "Software Requirements: Are They Really a Problem?" *Proc. Second Int'l Conf. on Software Eng.*, IEEE CS Press, 1976, pp. 61-68.

♦ B. Boehm, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, Jan. 1984, pp. 75-88.

♦ A. Davis, "A Taxonomy of the Early Stages of the Software Development Life Cycle," *J. Systems and Software*, Sept. 1988, pp. 297-311.

♦ A. Davis et al., "Identifying and Measuring Quality in Software Requirements Specifications," *Proc. IEEE Int'l Metrics Symp.*, IEEE CS Press, 1993, pp. 141-152.

♦ P. Hsia, A. Davis, and D. Kung, "Requirements Engineering: A Status Report," *IEEE Software*, Nov. 1993, pp. 75-79.

♦ M. Jaffe et al., "Software Requirements Analysis for Real-Time Control Systems," *IEEE Trans. on Software Eng.*, Mar. 1991, pp. 241-258.

♦ A. Levene and C. Mullery, "An Investigation of Requirements Specification Language: Theory and Practice," *Computer*, May 1982, pp. 50-59.

♦ G.-C. Roman, "A Taxonomy of Current Issues in Requirements Engineering," *Computer*, April 1985, pp. 14-23.

♦ W. Royce, "Software Requirements Analysis: Sizing and Costing," in *Practical Strategies for Developing Large-Scale Software*, E. Horowitz, ed., Addison-Wesley, 1975, pp. 57-71.

♦ R. Yeh and P. Zave, "Specifying Software Requirements," *Proc. IEEE*, Sept. 1980, pp. 1077-1085.

♦ R. Yeh, "Requirements Analysis – A Management Perspective," *Proc. Compsac*, IEEE CS Press, 1982, pp. 410-416.

Many of these and other excellent papers appear in Richard Thayer and Merlin Dorfman, *System and Software Requirements Engineering*, IEEE CS Press, 1990.

Alan M. Davis, University of Colorado at Colorado Springs

♦ Even when requirements are stated up front, it is likely they will change at least once during development, and it is certain they will change immediately after deployment.

♦ The time between the requirements phase and product delivery is usually too long to pinpoint how specific requirements-engineering techniques contributed to a product's success or failure.

CLASSIFIED RESEARCH

Requirements engineering has a fairly narrow goal – determine a need and define the external behavior of a solution – but the range of research into requirements is enormous. Merlin Roman provided the first classification of research topics nearly a decade ago.[1] We believe this number-letter classification scheme helps explain the areas the articles in this issue address.

e. Improvements to *eliciting* and understanding needs.

p. Improvements to documenting

the *problem*.

b. Improvements to describing the external *behavior* of the solution, including both functional and nonfunctional requirements.

q. Improvements to ensuring *quality* in the process or product.

m. Improvements to the *maintenance* and evolution of requirements.

Research will often address one area to improve a second. For example, prototyping research usually addresses how to describe external behavior, but its goal is to improve elicitation. This research would get the designation *be*, implying that *e* is the result of *b*.

Research in these categories can be further classified according to one of five areas:

P. Principles or rules that should always be true.

M. Step-by-step *methods* embodied in procedures, techniques, and process.

C. Conceptual models that define underlying semantic concepts (like finite-state machines).

L. Languages.

T. Tools.

Finally, research can be classified as to how soon the research expects its results to be applied to industrial practice:

10. Long-term research, more than 10 years out.

5. Midterm research, five to 10 years out.

2. Experimental "ripe fruit," ready for experimentation in industry, with more widespread use likely in two to four years.

0. "Ripe fruit," already proven (perhaps via joint development by researchers and the practitioners).

TRANSFERRING RESEARCH

Obviously, the ripened fruits of research are the easiest to transfer to industry, but at least four more roadblocks remain:

♦ Many researchers have little experience with industrial software development, so their proposed solutions often don't match real problems.[2]

♦ It is extremely difficult to conduct controlled experiments of software in a realistic setting, so techniques or tools are rarely backed up with solid statistics.[3] Most practitioners consider the results of smaller experiments insufficient "proof" that a new technology or tool is usable.

♦ Schedule pressures in industry make it difficult to "experiment" with technology that may be new to an organization.

♦ Market pressures make organizations reluctant to risk wasting precious resources on "unproven" technology.

EVEN THE RESEARCH THAT IS RIPE FOR USE IN INDUSTRY MAY NOT CLEAR ALL THE OBSTACLES IN THE ROAD.

With these issues in mind, the *IEEE Software* Editorial Board defined standards for conferences that aim to facilitate technology transfer in selected areas of software engineering.

The International Conference on Requirements Engineering, to be held in April in Colorado Springs, Colorado, is the first conference to be designated an *IEEE Software* Technology-Transfer Conference.

IN THIS ISSUE

Of the 86 papers submitted to ICRE, we have selected the best three to be featured in this issue. Preceding this group is a short essay by Jawed Siddiqi, who challenges some accepted notions about requirements engineering, to stimulate debate.

The first two articles describe work that falls into the *beqM2* category, but each emphasizes a different part of requirements engineering. Colin Potts and colleagues present the Inquiry Circle model, a structure that stakeholders can use to help clarify information needs – an especially important front-end activity. The model uses a conversation metaphor that encourages discussion about proposed requirements and helps make assumptions and rationales explicit. The authors apply the model to the requirements document of a meeting scheduler and show how scenario analysis – examining particular situations of expected behavior – is effective at surfacing new requirements and identifying inconsistencies in old ones.

Pei Hsia and colleagues also address scenario analysis, but from a more formal view, which is important to back-end activities like requirements validation. It is their belief that scenario analysis is not as widespread as it could be because there is no systematic way of applying it. The authors present a scenario-analysis model and a formal method and use both to analyze, generate, and validate scenarios for a simple PBX system. Although application is limited to systems that have a single response for a stimulus, the method is part of the groundwork for developing a complete scenario-analysis environment.

The authors also show how it supports acceptance testing – important feedback for requirements analysis – which usually cannot be done until after the system is built.

Odd Ivar Lindland and colleagues describe work that falls into the *qC2* category. Quality is an important part of any development activity, but it is critical to requirements engineering. It is hard enough to get what the customer wants with a good conceptual model. With a bad one it is impossible. The authors look at how others have defined quality goals in conceptual modeling and present their own framework, which addresses not only the goals but how to achieve them. While the framework has not been developed to the extent that it can be used in individual projects, it helps provide a better understanding of quality, which will inevitably lead to more satisfactory products.

In 1991, Davis and Peter Freeman characterized the state of research in requirements engineering as poor and recommended a forum be established to exchange ideas among requirements researchers and practitioners.[4] We hope ICRE will provide that forum and that this special issue brings some promising research to the attention of a much wider audience. ♦

ACKNOWLEDGMENTS

We thank the ICRE '94 program committee, who recommended papers for this issue, especially C. Shekaran, G. Zelesnik, C.G. Chung, J. Brackett, M. Alford, Merlin Dorfman, and Jawed Siddiqi.

REFERENCES

1. G.-C. Roman, "A Taxonomy of Current Issues in Requirements Engineering," *Computer*, Apr. 1985, pp. 14-23.
2. A. Davis, "Why Industry Often Says 'No Thanks' to Research," *IEEE Software*, Nov. 1992, pp. 97-99.
3. C. Potts, "Software Engineering Revisited," *IEEE Software*, Sept. 1993, pp. 19-28.
4. A. Davis and P. Freeman, "Requirements Engineering," *IEEE Trans. Software Eng.*, Mar. 1991, pp. 210-211.

Alan M. Davis is a professor of computer science and El Pomar chair of software engineering at the University of Colorado at Colorado Springs and at the Colorado Institute for Technology Transfer and Implementation. He is the author of *Software Requirements: Analysis and Specification* (Prentice-Hall, 1990) and the author or coauthor of more than 40 papers on software and requirements engineering. He is also the coeditor of *IEEE Software*'s Manager column with Winston Royce and associate editor of *Journal of Systems and Software* .

Davis holds a BS in mathematics from the State University of New York at Albany and an MS and a PhD in computer science from the University of Illinois. He is a senior member of the IEEE.

Pei Hsia is a professor of computer science engineering at the University of Texas at Arlington, and an associate editor-in-chief of *IEEE Software*. His research interests include requirements engineering, software-development paradigms, rapid prototyping, and software testing. His current focus is software incremental-delivery technology.

Hsia received a PhD in computer science from the University of Texas at Austin. He is a member of the ACM, Sigma Xi, and Phi Beta Delta and a senior member of the IEEE.

Address questions about this issue to Davis at University of Colorado, 1867 Austin Bluffs Pkwy., Ste. 200, PO Box 7150, Colorado Springs, CO 80933-7150; adavis@vivaldi.uccs.edu or to Hsia at Computer Science Engineering Dept., University of Texas, PO Box 19015, Arlington, TX 76019-0015; hsia@cse.uta.edu.