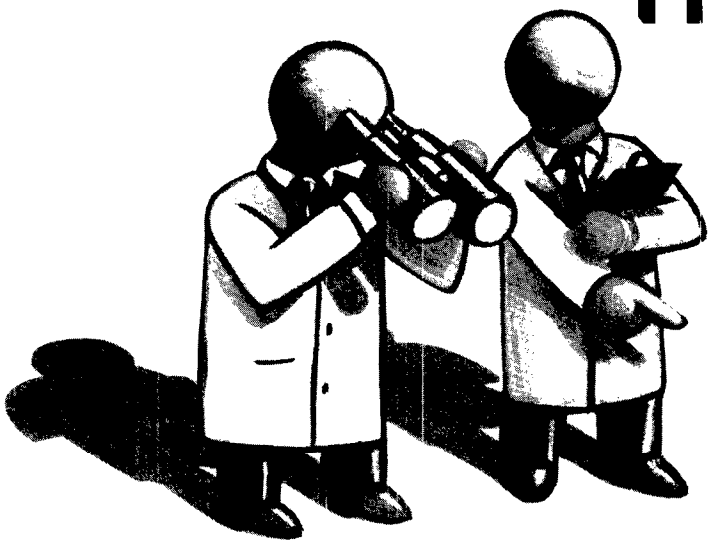
# Ecological Studies of Professional Programmers



*Guest Editors' Introduction:*

**BEN SHNEIDERMAN,**
*University of Maryland*
**JOHN M. CARROLL,**
*IBM Watson Research Center*

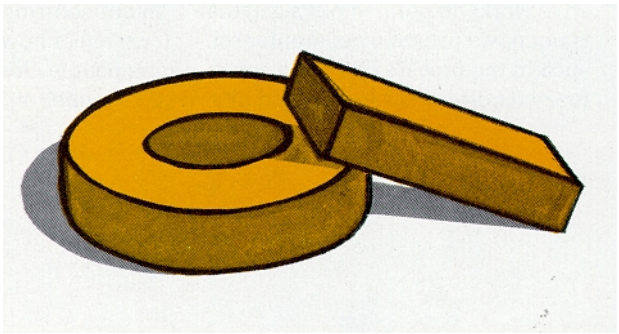
For over two decades, software psychology researchers have been developing insights to software productivity and quality by investigating builders and users of software. This research has been diverse in both its approach and its impacts. It has introduced systematic behavioral measurement into the software development process and into research on new software techniques and technologies, and has also opened up new social and cognitive interpretations of software processes [5, 12].

We now see evidence of a new thrust in software psychology coming to the fore, one in which usability researchers are direct participants in the definition and creation of new software artifacts. We call this paradigm Ecological Design, to emphasize (1) that realistic software situations are being confronted on their own terms, and (2) that the work is directed toward design results, not merely toward evaluation and description in the service of design goals.

The reorientation towards studying teamwork was prompted in 1971 by Weinberg and followed by a few researchers at that time, but the movement has accelerated with the recent and intense interest in computer supported collaborative work [15]. This was apparent in the papers presented at the two workshops on Empirical Studies of Programmers [10, 13]. An accompanying shift has also occurred in the software engineering community. The traditional waterfall model of software development with the precise specification of a provable topdown design is giving way to newer exploratory styles of program development that emphasize rapid prototyping and iterative refinement. The shift from product to process also puts greater emphasis on team organization, group processes, management policies, reusability, development tools, design methods, debugging strategies, and maintenance [6].

The three papers in this special section exemplify this new paradigm. Rosson, Maass, and Kellogg and Curtis, Krasner, and Iscoe describe highly qualitative studies of professional designers that produced specific technical proposals for improving software tools and the coordination of project management, an assessment of major bottlenecks, and a new framework for thinking about software design as a learning and communication process. Soloway, Pinto, Letovsky, Littman, and Lampert describe the design and exploration of software documentation that grew out of similarly qualitative studies of program maintenance.

We caution that this research paradigm is still in its infancy: setting design requirements and developing prototypes are not traditional activities of psychological researchers. These roles are still emerging, still being reconciled with the earlier paradigms. The particular projects highlighted here are only the beginning; the field continues to evolve, as more researchers are attracted, as more topics are explored, as more methods



are developed. Thus, despite the shortcomings of any particular project, the trajectory of this paradigm seems clear to us: it is the development of ideas that directly impact productivity and quality in software. Indeed, part of our intention in presenting this special section is to encourage more and more rapid development of the new paradigm.

## WORKING OUTSIDE THE LABORATORY

Software design takes place in software shops, not in psychological laboratories. To be pertinent, empirical work must confront design problems on their own terms: it must address whole problems while they are still technologically current and when their resolution can still constructively impact the direction of technological evolution. Software psychology in the paradigm of ecological design takes place in software shops.

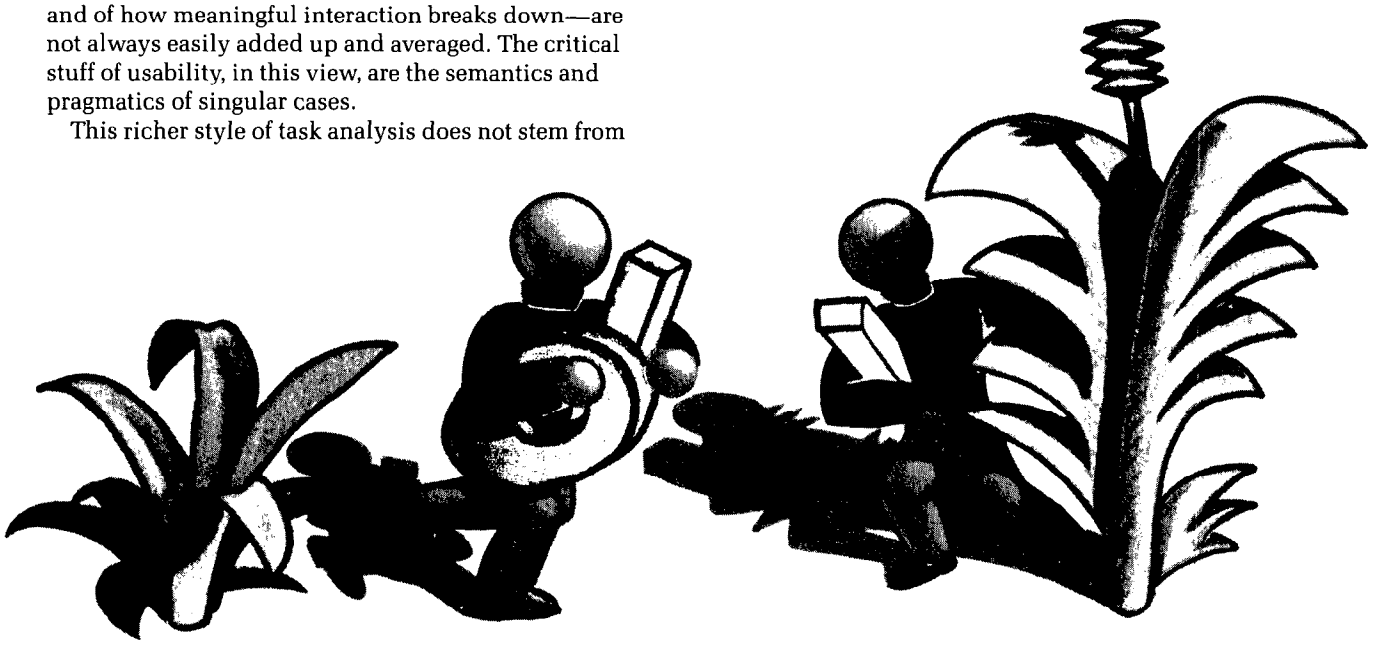
This work stresses richer analyses of software professionals working on realistic tasks. The main research setting for this work is the case study. A case study can begin and end anywhere in the task-artifact cycle. The key requirement is access to real situations. Case study task analysis usually consists of the collection of detailed, qualitative information (for example, thinking aloud protocols, interviews). Such data are arbitrarily rich: they can be returned to and analyzed from many different perspectives. The typical approach is to make videotapes to create a vivid and permanent data library.

There are some clear tradeoffs here. Rather than collect a few pieces of information (error rates, task times) about many individuals, this work seeks to collect lots of information from perhaps a small number of individuals. Studying unique situations at a fine grain of detail will generally not produce information suitable for conventional statistical analysis. Statistical approaches must assume a large population of comparable events. But the most important events in the use of software—the contextualized details of meaningful interaction, and of how meaningful interaction breaks down—are not always easily added up and averaged. The critical stuff of usability, in this view, are the semantics and pragmatics of singular cases.

This richer style of task analysis does not stem from a hypothesis testing procedure. It is interpretive, inductive; it seeks to discover, not merely to confirm or disconfirm. It typically involves collecting information over a significant span of time to eliminate ephemeral effects. Monitoring patterns of actual use of a software environment often supplement the more direct interview and protocol techniques.

Inasmuch as the chief goal of task analysis in the task-artifact cycle is to produce requirements for subsequent design work, this places emphasis on identifying big factors—big needs, big usability problems. Thus, one typical output of this phase is an error taxonomy, a qualitative theory of what is giving the user trouble, how it is happening, what users are doing as a consequence, etc. The complexity and rapid evolution of software technology requires richer and more open-ended methods than the one-off hypothesis testing of the human factors evaluations and cognitive description approaches.

The papers by Curtis et al. and Rosson et al. present analyses of interviews with software designers. The authors collected voluminous transcriptions characterizing design activity that had taken place over substantial spans of time. They sought to describe both the general patterns and the significant, but more singular experiences of particular designers and design teams. Curtis et al. offer an excellent example of the new style of analysis and deliver appealing insights about professional software design derived from 97 interviews with participants in 17 large projects. They apply five levels of analysis (individual, team, project, company, and business milieu) to three issues (the thin spread of application knowledge, fluctuating and conflicting requirements, and communications mechanisms and breakdowns). Rosson et al. describe interviews with 22 professional user interface designers and their projects. Several important issues are covered



including user testing and iterative design, the rising importance of user interface management systems and support tools, and the sources of ideas for designers. The key goal of both analyses was to develop empirical taxonomies of the big factors that structure software design activities and hence the outcomes of design.

## PARTICIPATING IN SOFTWARE DESIGN

Building and inventing artifacts is not usually viewed as a normal activity in psychological research [3, 7]. Yet, what is needed in software is psychologically informed artifacts. Modern software technology is brimming with psychological content (consider any discussion of the rationales for structured programming or object-oriented programming). The challenge is to describe more effective ways to infuse and systematize this psychological content in software design.

Carroll and Campbell [4] outline a research strategy that can be seen in various emerging ecological design projects. They introduce the notion of the "task-artifact cycle": psychological analysis of the tasks people want to perform, coupled with the problems, insights, and satisfactions they experience, followed by setting the objectives for new software tools, constrained by technological feasibility. New tools, in turn, alter the tasks for which they were designed, indeed, alter the situations in which the tasks occur and even the conditions that cause people to want to engage in the tasks. This creates a need for further task analysis, and in time, for the design of further artifacts.

While ecological design is still an emerging approach, one can identify projects that illustrate some of the major themes, that is, projects in which psychological rationales are driving the design of new software artifacts for real situations. Key to all these projects is the attempt to integrate psychological task analysis of real work situations with the development of new software artifacts to improve usability, productivity, and satisfaction.

Soloway and colleagues, working at the NASA/Jet Propulsion Laboratory, have completed several cycles of task analysis/artifact design in the area of software documentation; psychological analyses of the role and use of documentation in software maintenance were the driving forces behind the design of several different forms of documentation (see paper this issue, and [8], [9], [14]). For example, they found that a typical strategy of professional programmers for using the documentation (the as-needed strategy) often resulted in subjects missing key information about the non-local interactions in the program; this observation drove the design of a version of the documentation that attempted to make that key information more readily accessible.

Anderson and his collaborators have codified their analyses of how students learn to program in Lisp [1] into the design of tutoring systems for teaching Lisp [2, 11]. The tutor is now in regular use in several Lisp curricula, and its performance compares quite favorably with more standard lecture methods of instruction.

We see the emergence of a more proactive role for cognitive and social science in the invention of new software technology as critically important to the more traditional goals of software psychology, namely, facilitating productivity and quality in software, and developing more powerful and conceptually interesting theoretical models. We see the possibility of this role as deriving both from the genuine research successes in software psychology and from the two decades of practical experience that software psychologists have attained. Software technology is complex and fragmented. To play a directive role in the development of these technologies, one must be intimately and broadly involved; ideas come from the informed confrontation of what is with what should be/what needs to be. It is our sense that the field of software psychology has matured to a point where it can play a leading role in this invention/exploration process.

The papers presented here have been refined through a highly collaborative and intense reviewing process. We gratefully acknowledge the important role played by the reviewers in generating an excellent set of papers.

## REFERENCES


1. Anderson, J. T., Farrell, R., and Sauers, R. Learning to program in Lisp. *Cognitive Sci. 8,* (1984) 87–129.
2. Anderson, J. R., and Skwarecki, E. The automated tutoring of introductory computer programming. *Commun. ACM 29,* 9 (Sept. 1986), 842–849.
3. Carroll, J. M. *Evaluation, Description and Invention: Paradigms for Human-Computer Interaction.* Vol. 28, *Advances in Computers.* Academic Press, New York, 1988.
4. Carroll, J. M., and Campbell, R. L. Artifacts as psychological theories: The case of human-computer interaction. Res. Rep. RC 13454. IBM, Yorktown Heights, N.Y., 1988. To be published in *Behav. Inf. Tech.*
5. Curtis, B., Soloway, E., Brooks, R., Black, J., Ehrlich, K., and Ramsey, H. R. Software psychology: The need for an interdisciplinary program. *Proc. IEEE 74,* 8 (Aug. 1986), 1092–1106.
6. Floyd, C. Outline of a paradigm change in software engineering. In *Computers and Democracy: A Scandinavian Challenge,* G. Bjerknes, P. Ehn, and M. Kyng Eds. Averbury-Grower Publishing, Brookfield, Ver., 1987, pp. 191–210.
7. Landauer, T. K. Psychology as a mother of invention. In *Proceedings of CHI + GI '87: Human Factors in Computing Systems and Graphics Interface* (Toronto, April 5–9), ACM, New York, 1987, pp. 333–335.
8. Letovsky, S., and Soloway, E. Delocalized plans and program comprehension. *IEEE Softw. 3,* 3 (May 1986), 41-49.
9. Littman, D., Pinto, J., Letovsky, S., and Soloway, E. Mental models and software maintenance. In *Empirical Studies of Programmers,* E. Soloway and S. Iyengar, Eds. Ablex, Norwood, N.J., 1986, pp. 80-98.
10. Olson, G., Sheppard, S., and Soloway, E. *Empirical Studies of Programmers: Second Workshop.* Ablex, Norwood, N.J., 1987.
11. Reiser, B. J., Friedman, P., Gevins, J., Kimberg, D. Y., Ranney, M., and Romero, A. A graphical programming language interface for an intelligent Lisp tutor. CSL Rep. 15. Princeton Univ., N.J., 1988.
12. Shneiderman, B. *Software Psychology: Human Factors in Computer and Information Systems.* Little Brown and Co., Boston, Mass., 1980.
13. Soloway, E., and Iyengar, S. *Empirical Studies of Programmers.* Ablex, Norwood, N.J., 1986.
14. Soloway, E., Pinto, J., Fertig, S., Letovsky, S., Lampert, R., Littman, D., and Ewing, K. Studying software documentation from a cognitive perspective: a status report. In *Proceedings of the 10th Annual NASA/Goddard Software Engineering Workshop* (November, Greenbelt, Md.), Greenbelt, Md., 1986.
15. Weinberg, G. *The Psychology of Computer Programming.* Van Nostrand Reinhold, New York, 1971.