Building Bridges and Interfaces: Toward the Next Generation of UIMS

H. Rex Hartson
Deborah Hix

TR 87-3

Building Bridges and Interfaces:

Toward the Next Generation of UIMS

H. Rex Hartson (Contact Author)
Deborah Hix

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg VA 24061
703/961-4857

ABSTRACT: User interface management systems (UIMS) have established themselves in both research and commercial arenas. We present several generations in UIMS evolution and discuss some problems of the early generations. In particular, we discuss the problems of a gap between methods used by behavioral scientists and computer scientists during the process of building interfaces. We present an empirical approach to begin bridging this gap and results of our preliminary observations: a human-computer interface development life cycle and recording techniques for interface development, as well as UIMS needed to support them. We conclude with future directions for the evolution of UIMS.

Number of words: approximately 2890 (excluding outline and references)

Keywords: UIMS, user interface management, human-computer interaction, human-computer interface, interface development methodology, interface development life cycle.

Topic area: User interface tool kits

Building Bridges and Interfaces:

Toward the Next Generation of UIMS

H. Rex Hartson
Deborah Hix

1. GENERATIONS OF UIMS

Unknown a few years ago, *user interface management systems* (UIMS) have become an exciting area of research and practice. Yet there are already signs of promises unfulfilled, due to a lack of both functionality and usability. We perceive a trend in UIMS evolution that we have somewhat arbitrarily divided into generations:

First generation:	Facade and prototype builders, predecessors to real UIMS.
Second generation:	UIMS with run time support and increasing design time and rapid prototyping tools, often with limited functionality and usability.
Third generation:	UIMS with increased functionality and flexibility through such advances as object orientation, direct manipulation, asynchronous and complex graphical dialogue; still a usability gap.
Fourth generation:	UIMS with improved usability through empirically-based refinement of both UIMS and interface development methodologies.
Future generations:	UIMS supplemented by artificial intelligence, such as expert systems for aiding interface development and knowledge-based processing.

UIMS of the first two generations have established themselves in both research and commercial arenas. Although first generation UIMS were mostly facade generators and prototypers (e.g., [Hanau and Lenorovitz, 1980; Mason and Carey, 1981; Wong and Reid, 1982]), they saw significant application in commercial development environments. Interfaces produced by these UIMS were typically specified by BNF-style languages, supplemented with conventional programming. Early second generation UIMS research often focused on support for interface execution [Guedj et al., 1980; GIIT, 1983], with little emphasis on the end-user, human factors, or interface design. This generation produced several experimental systems that contributed greatly to the knowledge and experience base [Wasserman, 1981; Kasik, 1982; Buxton et al., 1983; Kamran and Feldman, 1983; Olsen, 1983; Hartson, Hix, and Ehrich, 1984; Granor and Badler, 1986]. Second generation UIMS have been commercially available for some time [e.g., Rubel, 1982; Schulert, Rogers, and Hamilton, 1985; Wasserman and Shewmake, 1985]. Second generation UIMS varied greatly in their capabilities and usability, with

many requiring conventional programming. Moving toward the third generation, the UIMS view has broadened [Tanner and Buxton, 1984; Olsen et al., 1984; Hartson and Hix, 1987] to include emphasis on the end-user and the non-programming interface developer.

Third generation UIMS are adding functionality to address specific problems. For example, limitations of sequential dialogue are overcome by event-based asynchronous dialogue [Green, 1985] and by an order independent form, or slot, abstraction [Hayes, 1985]. Object orientation allows greater flexibility and ease of implementation [Sibert, Hurley, and Bleser, 1987]. Dialogue techniques can be specified by demonstration [Myers, 1987]. While the third generation is producing interesting new UIMS ideas, it has not been the needed empirical refinement of the previous generations.

The premise that UIMS improve user interface quality and shorten development time is generally accepted, despite a scarcity of empirical evidence to this effect, and despite their sometimes limited scope and difficulty of use. We believe the state-of-the-art in UIMS evolution is into the third generation, and we see a need to produce a next generation of more usable UIMS, complementary to improvements in interface development methodology. Much research has addressed end-users; now it is time to address interface developers --- as users of UIMS. Those who have begun work in this direction include Mantei [1986], Carey [1987], and Rosson, Maass, and Kellogg [1987].

The work we report here is an embryonic observational approach toward what Rosson, Maass, and Kellogg term "designing for designers". This paper presents some problems of early generation UIMS; an empirical approach to begin solving these problems; and results of the observations: a human-computer interface development life cycle and recording techniques, as well as capabilities of UIMS needed to support them. It concludes with future directions for the evolution of UIMS.

2. PROBLEMS WITH EARLY GENERATION UIMS: A Gap to be Bridged

Cooperation between behavioral scientists and computer scientists is necessary for development of interactive computer systems with quality human interfaces [Hartson, 1985]. However, the problem is a gap caused by differences

between methods used by behavioral scientists to design and analyze interfaces and those used by computer scientists to design and implement software. Mechanical aspects of developing interfaces (and especially of using tools) hamper performance of the task itself; the syntactic domain interferes with the semantic domain [Shneiderman and Mayer, 1979]. Particularly with the increased functionality of third generation UIMS, an interface developer can easily lose sight of the interface development task, getting lost in the UIMS itself. The gap is manifest in the lack of, first, a *holistic development methodology* and, second, usable *interactive tools* for supporting activities of this broader methodology.

First, consider the methodology problem. No accepted methodology exists to guide interface development. The few interface development approaches that do exist lack connections to software engineering, thus widening the gap. Wasserman's User Software Engineering (USE) methodology [Wasserman, 1981] and the Dialogue Management System (DMS) methodology [Yunten and Hartson, 1985] come closest to providing a connection to software engineering. Second, consider the tool problem. As with the methodology, most UIMS (exceptions again include USE and DMS) do not make a connection with software engineering. Current UIMS typically support only a small subset of the overall interface life cycle activities. Furthermore, few observational data exist of developers performing development activities (cf. [Hammond et al., 1983; Gould and Lewis, 1985; Lammers, 1986; Rosson, Maass, and Kellogg, 1987]). Thus, the limited existing methodologies and tools for interface development have typically evolved based on best guesses and intuition.

3. APPROACH TO SOLVING THE PROBLEMS: Empirical Observations

In an effort to begin bridging the gap, we conducted several studies, not to formally test a hypothesis, but rather to investigate natural --- i.e., without *a priori* bias from specific approaches, notations, and tools --- and effective ways in which quality interfaces are developed. Our approach follows the principles of system design recommended by Gould and Lewis [1985]: focus on users (of UIMS) early in the design process, empirical studies throughout all phases of system evolution, and iterative refinement based on the empirical results.

We conducted three protocol-gathering case studies involving interface developer subjects producing different kinds of interactive systems. In the most extensive of these, three subjects developed (including full implementation) a document retrieval system over a two year period. This system was chosen because it represented a broad variety of interface styles and techniques (e.g., menus, function keys, typed text input), interface features (e.g., direct manipulation), and included both sequential and multi-thread dialogue. Subjects were computer science students who performed this as project work for course credit†. Two subjects had several years of industrial experience in software development environments. Subjects were given a written functional description of application system semantics, with very little information about the interface and no specific methodological instruction about interface design.

Although details varied, the following describes what typically happened. All three subjects used similar approaches, starting with pencil-and-paper scenarios of what the end-user sees and does (e.g., viewing displays, pressing keys, entering commands, using a mouse). They initially represented the design with a set of numbered sketches of screen displays, writing in information about sequencing among screens. To show transitions among screens graphically, they developed representations varying from flow charts to state diagrams that were detailed and concrete, yet large and complex. This pencil-and-paper "prototype" was tested manually, and changes immediately fed back to requirements specifications.

Next, subjects alternated from these bottom-up scenarios and state diagrams to a top-down approach, working downward through the representation, analyzing, structuring, and modifying the design, until they could implement a first version of the document retrieval system. Then they alternated back to a bottom-up step of end-user testing to refine the interface iteratively, focusing primarily on interface details. Before end-users were reasonably satisfied with quality and functionality, developer subjects had gone through three complete revision cycles, and many iterations for fine-tuning interface details.

---

† Studies by Holt, Boehm-Davis, and Schultz [1987] indicate that use of students for subjects in software and methodological studies is an acceptable practice, yielding useful results.

In two other similar, but less extensive studies, subjects produced some form of scenarios and state diagrams as a starting point for interface development. Added support has been given to our observations by two years of co-operative research with IBM Federal Systems Division. We found that much of their interface development, at least informally, is scenario-driven. In addition, Carey [1987] relates observations of numerous independent design teams; all produced pencil-and-paper versions of screen sequences early in the development process. Finally, the interface developers' life cycle and representational needs we observed are consistent with Piaget's [1950] theory that people naturally learn by starting with concrete examples and working toward the abstract.

4. *RESULTS OF THE OBSERVATIONS: A Life Cycle, Recording Techniques, and Next Generation UIMS for Interface Development*

It is probably impossible to determine empirically (or otherwise) the "best" way to develop interfaces. But our observations have yielded indications of procedural (life cycle) and representational (recording) needs of developers, as well as tool requirements.

We conclude that a unified *holistic methodology* is needed to treat human-computer interface development as an integral part of the software engineering process. It consists of (1) a *life cycle* for interface development and (2) a set of *recording techniques* for capturing outputs of each phase of this life cycle. A *new generation of UIMS* is also needed to support all phases of the life cycle, including task analysis, requirements, and early creative interface design activities --- that tight mental loop of synthesis and analysis involving dialogue scenarios, creative doodling, and experimentation with ideas. Without this support, interfaces produced with UIMS will not necessarily be better than those produced without UIMS. In fact, early generation UIMS may serve only as a means for faster production of bad interfaces.

4.1. *A Life Cycle for Interface Development*

Bridging the gap in the interface development process necessitates a non-conventional life cycle. Two relevant results are discussed below:

- *We hypothesize, based on our observations, that interface development occurs in "alternating waves" of bottom-up and top-down activities.*

- *We are evolving an interface development life cycle that supports this hypothesis.*

Evolution of Our Hypothesis. Draper and Norman [1985] draw a parallelism between interface development and software engineering. Now that some researchers [Swartout and Balzer, 1982; Ramamoorthy et al., 1984] are beginning to see that a linear life cycle is not the most appropriate for all software, the arguments of Draper and Norman are more compelling. Our observations have led us to conclude that the life cycle for interface development does not "naturally" follow the traditional software development life cycle, with its top-down linear sequence of requirements, design, implementation, and testing. In fact, attempts to impose the classical "waterfall" paradigm on interface development are undoubtedly the cause of many bad interfaces. Rather, we hypothesize that the interface development life cycle most naturally occurs in "alternating waves" of two kinds of activities. Typical early phases of interface development are bottom-up, based on concrete dialogue scenarios, often augmented with state-diagram-like representations of sequencing. Subsequent phases involve top-down, step-wise decomposition and structuring. Activities that are bottom-up, synthetic, empirical, and reflect the end-user's view alternate with activities that are top-down, analytic, structuring, and reflect the system's view. These two kinds of development activities reflect different kinds of mental modes, which we call "Mode A" and "Mode B", summarized below.

Mode A	*Mode B*
Top-down	Bottom-up
Analytic	Synthetic
"Stop"	"Go"
Organizing	Free-thinking
Judicial	Creative
Structural	Behavioral
General	Detailed
Modeling or formal	Empirical or *ad hoc*
Reflects system's view and works toward end-user	Reflect's end-user's view and works toward system
Linguistic orientation	Dialogue flow orientation

Of course, development of interfaces and software requires both these mental modes. The top-down nature of mode A activity is more useful when the developer has some experience and *a priori* knowledge of target system structure. The bottom-up nature of mode B activity is more suitable in novel design situations, when little about target system structure is known in advance, and "trial and error" augment experience and intuition to develop system structure. As interface development knowledge and experience are accumulated, top-down techniques will be used more.

An Interface Development Life Cycle. We have extrapolated our observations into a cyclical model of the interface development life cycle, shown simplistically in Figure 1.

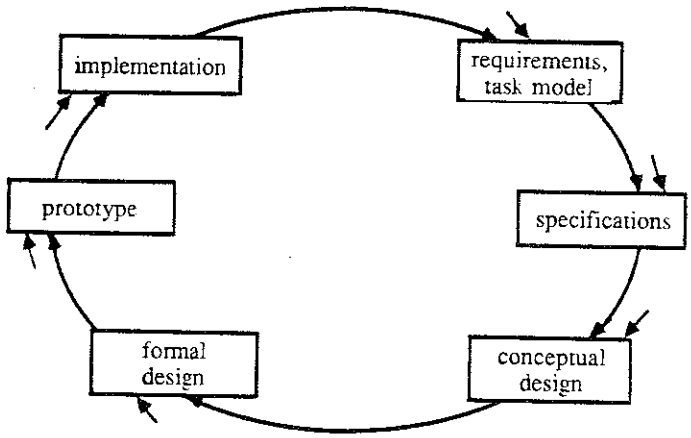

Figure 1. An Interface Development Life Cycle

This is the kind of life cycle we foresee being supported by next generation UIMS†. Because of its totally cyclic nature, almost any place (except possibly implementation) is appropriate to enter this life cycle. Thus, a later entry point (e.g., a conceptual design activity such as playing with scenarios) may be followed by an "earlier" activity (e.g., formulation of requirements

† There are obviously similarities between this figure and other system life cycles. However, only a few other researchers address interface design methodology (e.g., [Wasserman et al., 1986]).

specifications). This again opposes traditional top-down paradigms, but fits our hypothesis of alternating waves of development.

4.2. Recording Techniques for Interface Development

An important part of bridging the gap in the interface development process is *communication*. The key to this communication is *recording techniques* --- unambiguous, easy-to-produce mechanisms to convey specification and design of the evolving interface to all developer roles. Two relevant results are discussed below:

- *Based on our observations, we know something about the nature of the interface recording problem (what to record).*
- *We are developing recording techniques to address those aspects of interfaces we know must be recorded (how to record).*

Nature of the Recording Problem. At least three issues are involved in assessing the nature of the recording process. First, our observations showed that different interface developer roles need a common representation for communicating, but have different needs for representation in their own development work. One solution is techniques that allow a common view, but to which *filters* can be applied to produce different working views for each role. Maintaining multiple views manually (e.g., pencil and paper) is difficult, but computer-based tools such as UIMS can maintain a single representation from which various views are derived on demand.

The second issue relates to the need for matching mental and physical activities of interface developers. Producing representations of the interface is a physical task that is necessarily preceded by a corresponding mental task to create the design. This sequence of mental and physical activity is illustrated by the boxes labelled respectively "conceptual design" and "formal design" in Figure 1. With current UIMS, the mental and physical tasks tend to be quite separate, causing discontinuity in an interface developer's work. Thus, another result of our observations is that mental and physical interface development tasks must be brought closer together by providing interactive recording tools (UIMS) that closely match the mental processes they physically support.

The third issue is the magnitude of the unsolved problem of providing a complete, consistent, and readable record of an interface design. Complete description of an interface requires massive amounts of complex details, including behavioral and structural, visible and non-visible aspects that UIMS must be able to capture.

A Set of Recording Techniques. Based on our observations, we are developing recording techniques to support our alternating waves approach. The techniques are called *augmented scenarios*, which support the bottom-up, behavioral, end-user-oriented mode of development; and *augmented supervised flow diagrams*, which support the top-down, structuring, system-oriented mode of development and provide filters for different views of the design [Hartson, Hix, and Kraly, 1987]. Both techniques are variations of the formal concept of a state machine, giving our approach a firm theoretical basis. We are also producing formal methods for mapping from one technique to another. Space does not permit further elaboration here; see [Hartson, Hix, and Kraly, 1987] for details and examples.

4.3. Interactive Tools to Support Interface Development: The Next Generation of UIMS

Another key to bridging the interface development gap are UIMS that support the life cycle and recording techniques of the holistic methodology. Two relevant results are discussed below:

- *Based on our observations, tools should support both bottom-up and top-down activities.*

- *Tools should be empirically derived, formally evaluated, and iteratively refined.*

Support for Alternating Waves. It is evident from our observations that next generation UIMS must support all activities of the life cycle and allow interactive recording of interface designs. In particular, UIMS to support the alternating waves approach must allow scenarios as a starting point for development. They must allow the developer immediately to observe any part of interface form, content, and sequencing behavior already defined, without being hindered by incomplete and missing parts (a severe problem with most existing

computer-based tools). UIMS must make visible an end-user-oriented view of interface behavior and provide rapid modifiability throughout the life cycle.

Evolution of Next Generation UIMS. To improve UIMS usability, the interface to the UIMS itself must be given considerable attention, so that it provides the best possible support for an interface developer. This is achievable only through empirical evaluation and iterative refinement of existing UIMS, a much neglected aspect of UIMS development thus far. Approaches for empirically-based tool development are suggested in Section 5.

5. FUTURE DIRECTIONS IN UIMS RESEARCH

Our work reported here, although preliminary, is some of the first in which experimental studies and their qualitative results have been used to drive the development of methodology and tools for human-computer interface management. We see the need for longer term, formal observations in "real world" software engineering project environments, alternating with tool development driven by these observations. Within our DMS project we are continuing this initial work, including plans for extensive protocol-gathering from interface developers, with emphasis on early life cycle activities. Analysis of these observations will increase our understanding of recording needs and allow us to refine our corresponding methodological and tool support.

We believe that such an empirical approach to "designing for designers" constitutes a profitable avenue for all researchers in this area of human-computer interface management. For far too long, the need for specialized life cycles and recording techniques tailored to interface development has been ignored, and tools have been produced based solely on their creators' intuition and best guesses. It is time to go to the source --- the interface developers themselves --- to gather empirical evidence about their needs for building interfaces. Such activities will result in interface development techniques and tools that are empirically, rather than blindly, derived. This will help bridge the gap among all interface developer roles, bringing them together into the same interactive system development process. Building such bridges will take us to the next generation of UIMS and ultimately help us to build better interfaces.

ACKNOWLEDGMENTS

We wish to acknowledge the participation in our Dialogue Management Project research by our colleague and friend, Dr. Roger W. Ehrich, as well as research assistants Eric Smith, Antonio Siochi, Matt Humphrey, and Jeff Brandenburg. Mr. Tom Kraly of IBM FSD contributed to the interface recording techniques research. JoAnne Lee Bogner's speedy typing was invaluable. This research is funded by the National Science Foundation, Dr. H. E. Bramford, Jr., Program Director; IBM Federal Systems Division; the Software Productivity Consortium; and the Virginia Center for Innovative Technology.

REFERENCES

Buxton, W.A., Lamb, M.R., Sherman, D., and Smith, K.C. 1983. Towards a Comprehensive User Interface Management System. *Computer Graphics, 17* 3 (July), 35-42.

Carey, T. The Gift of Good Design Tools. 1987. To appear in *Advances in Human-Computer Interaction, Volume 2.* (H.R. Hartson and D. Hix, eds.) Ablex Publishing Corp.

Draper, S.W. and Norman, D.A. 1985. Software Engineering for User Interfaces. *IEEE Transactions on Software Engineering, SE-11.* 252-258.

(GIIT) Graphical Input Interaction Technique Workshop Summary. 1983. *Computer Graphics, 17* 1 (January), 5-30.

Gould, J.D. ,and Lewis, C. 1985. Designing for Usability: Key Principles and What Designers Think. *Communications of the ACM, 28* 3, 300-311.

Granor, T.E., and Badler, N.I. 1986. GUIDE: Graphical User Interface Development Environment. In *Proceedings of Trends and Applications.* Silver Spring, MD, 37-41.

Green, M. 1985. The University of Alberta User Interface Management System. *Computer Graphics, 19* 3 (July), 205-213.

Guedj, R.A., ten Hagen, P.J.W., Hopgood, F.R.A., Tucker, H.A., and Duce, D.A., eds. 1980. *Methodology of Interaction: Seillac II.* Seillac, France, 1979. North-Holland.

Hammond, N., Jorgensen, A., MacLean, A., Barnard, P. and Long, J. 1983. Design Practice and Interface Usability: Evidence from Interviews with Designers. In *Proceedings of CHI '83 Conference.* Boston (December), 40-44.

Hanau, P.R., and Lenorovitz, D.R. 1980. Prototyping and Simulation Tools for User/Computer Dialogue Design. *Computer Graphics, 14* 3 (July), 271-278.

Hartson, H.R., ed. 1985. *Advances in Human-Computer Interaction, Volume 1.* Ablex Publishing Co.

Hartson, H.R., and Hix, D. Human-Computer Interface Development: Concepts and Systems for Its Management. To appear in *ACM Computing Surveys* in 1987.

Hartson, H.R., (Johnson), D. Hix, and Ehrich, R.W. 1984. A Human-Computer Dialogue Management System. In *Proceedings of INTERACT '84, First IFIP Conference on Human-Computer Interaction.* London (September), 57-61.

Hartson, H.R., Hix, D., and Kraly, T.M. 1987. Dialogue Management as an Integral Part of Software Engineering: Final Report. Virginia Tech Department of Computer Science Technical Report Number 87-22, (June).

Hayes, P. 1985. Executable Interface Definitions Using Form-Based Interface Abstractions. In *Advances in Human-Computer Interaction, Volume 1.* (H. Rex Hartson, ed.) Ablex Publishing Corp., 161-190.

Holt, R.W., Boehm-Davis, D.A., and Schultz, A.C. 1987. Mental Representations of Programs for Student and Professional Programmers. George Mason University Department of Psychology Technical Report.

Kamran, A. and Feldman, M.B. 1983. Graphics Programming Independent of Interaction Techniques and Styles. *Computer Graphics, 17* 1 (January), 58-66.

Kasik, D.J. 1982. A User Interface Management System. *Computer Graphics, 16* 3 (July), 99-106.

Lammers, S. 1986. *Programmers at Work.* Microsoft Press.

Mantei, Marilyn. 1986. Private communication with authors.

Mason, R.E.A., and Carey, T.T. 1981. Productivity Experiences with a Scenario Tool. In *Proceedings of the IEEE COMPCON.* Washington, D.C. (September), 106-111.

Myers, B. 1987. Creating Dynamic Interaction Techniques by Demonstration. In *Proceedings of CHI + GI '87 Conference.* Toronto (April), 271-277.

Olsen, D.R., Jr. 1983. Automatic Generation of Interactive Systems. *Computer Graphics, 17* 1 (January), 53-57.

Olsen, D.R., Jr., Buxton, W., Ehrich, R.W., Kasik, D.J., Rhyne, J.R., and Sibert, J. 1984. A Context for User Interface Management. *IEEE Computer* (December), 33-42.

Piaget, J. 1950. *The Psychology of Intelligence.* Harcourt, Brace.

Ramamoorthy, C.V., Prakish, A., Tsai, W.T., and Usuda, Y. 1984. Software Engineering: Problems and Perspectives. *IEEE Computer, 17* 10 (October), 191-209.

Rosson, M.B., Maass, S., and Kellogg, W.A. 1987. Designing for Designers: An Analysis of Design Practice in the Real World. In *Proceedings of CHI + GI '87 Conference.* Toronto (April), 137-142.

Rubel, A. 1982. Graphic Based Applications --- Tools to Fill the Software Gap. *Digital Design*, (July).

Schulert, A.J., Rogers, G.T., and Hamilton, J.A. 1985. ADM --- A Dialogue Manager. In *Proceedings of CHI '85 Conference*. San Francisco (April), 177-183.

Shneiderman, B. and Mayer, R. 1979. Syntactic/Semantic Interactions in Programming Behavior: A Model and Experimental Results. *International Journal of Computer and Information Sciences, 8* 3, 219-239.

Sibert, J.L., Hurley, W.D., and Bleser, T.W. 1987. Design and Implementation of an Object-Oriented User Interface Management System. To appear in *Advances in Human-Computer Interaction, Volume 2*. (H.R. Hartson and D. Hix, eds.) Ablex Publishing Corp.

Swartout, W., and Balzer, R. 1982. On the Inevitable Intertwining of Specification and Implementation. *Communications of the ACM, 25* 7 (July), 438-445.

Tanner, P.P., and Buxton, W.A. 1984. Some Issues in Future User Interface Management System (UIMS) Development. In *Seeheim Workshop of User Interface Management Systems*. Eurograhics-Springer.

Wasserman, A.I. 1981. User Software Engineering and the Design of Interactive Systems. In *Proceedings of the Fifth International Conference of Software Engineering*. 387-393.

Wasserman, A.I. and Shewmake, D.T. 1985. The Role of Prototypes in the User Software Engineering Methodology. In *Advances in Human-Computer Interaction, Volume 1*, (H. Rex Hartson, ed.) Ablex Publishing Corp., 191-210.

Wasserman, A.I., Pircher, P.A., Shewmake, D.T., and Kersten, M.L. 1986. Developing Interactive Information Systems with the User Software Engineering Methodology. *IEEE Transactions on Software Engineering, SE-12* 2 (February), 326-345.

Wong, P.C.S., and Reid, E.R. 1982. FLAIR -- User Interface Dialog Design Tool. *Computer Graphics, 16* 3 (July), 87-98.

Yunten, T., and Hartson, H.R. 1985. A SUPERvisory Methodology And Notation (SUPERMAN) for Human-Computer System Development. In *Advances in Human-Computer Interaction, Volume 1*. (H. Rex Hartson, ed.) Ablex Publishing Corp., 243-281.