UIMS: Toward the Next Generation

H. Rex Hartson
Deborah Hix

TR 86-41

UIMS:
Toward the Next Generation

H. Rex Hartson
Deborah Hix

1. *THREE GENERATIONS OF UIMS*

"This is the year of the UIMS." [Card, 1986]

And indeed these words rang true at the CHI '86 conference. *User interface management systems — UIMS* — were a major focus of many of the technical presentations, live demonstrations, and special interest sessions. But what does the future hold for UIMS? Will they be a passing fad or a viable tool? We believe they are here to stay. In fact, we envision at least three generations of UIMS:

First generation:	UIMS as they are now, with run time support and ever-increasing design time and rapid prototyping tools.
Second generation:	UIMS extended into early phases of the system development life cycle that they currently do not support, such as task analysis, requirements specification, and conceptual interface design, resulting in more powerful and more extensively usable UIMS.
Third generation:	UIMS pervaded by artificial intelligence, such as expert systems for aiding interface development, and natural language processing (both in the UIMS themselves and in their product systems).

First generation UIMS have established themselves both in the research and commercial arenas. Possibly not since the emergence of database management in the early 1970's has an area of computer science received so much attention as interface management. Early UIMS research focused on support for interface execution [Guedj, ten Hagen, Hopgood, Tucker, and Duce, 1980; GIIT, 1983], with little emphasis on the end-user, human factors, or interface design. Early UIMS-produced interfaces were typically specified by BNF-style languages or, more often, by coding the interface in a programming language. Thus, early first generation UIMS were tools for programmers. More recently, the UIMS view has broadened [Tanner and Buxton, 1984; Olsen, Buxton, Ehrich, Kasik, Rhyne, and Sibert, 1984; Hartson and Hix, 1987] to include emphasis on the end-user and the non-programming interface designer. First generation experimental UIMS vary greatly in their capabilities [e.g., Wasserman, 1981; Kasik, 1982; Buxton, Lamb, Sherman, and Smith, 1983; Olsen, 1983; Hartson, Hix, and Ehrich, 1984; Green, 1985; Hayes, 1985; Sibert, 1987]. Some systems for prototyping the human interface are closely related to UIMS [e.g., Hanau and Lenorovitz, 1980; Mason and Carey, 1981; Wong and Reid, 1982; Flanagan, Lenorovitz, Stamke, and Stocker, 1985]. First generation UIMS are now commercially available [e.g., Rubel, 1982; Schulert, Rogers, and Hamilton, 1985]. Despite this prevalence, UIMS evaluations are scarce. There seems to be, however, a generally accepted premise that UIMS are viable, despite their sometimes limited scope and difficulty of use. We envision the improvement of UIMS usability and the extension of UIMS to include a broader whole system development life cycle as the basis for evolution of a second generation of UIMS. Most previous research has addressed end-users; now it is time to address interface designers (as end-users of UIMS). Others who have begun work in this direction include Mantei [1986], with empirical foundations for improving UIMS usability, and Carey [1987], concerned with matching tools such as UIMS to their users and environment.

The remainder of this paper presents some problems of first generation UIMS in more detail, some informal empirical work that is leading toward an interface development life cycle and new UIMS to support it, and future directions for the anticipated evolution.

2. PROBLEMS WITH FIRST GENERATION UIMS

Cooperation between behavioral scientists and computer scientists is becoming accepted as necessary for successful development of interactive computer systems with quality human interfaces [Hartson, 1985]. However, there is a gap caused by differences between the methods used by behavioral scientists to design and analyze interfaces and those used by computer scientists to design and implement software. That gap inhibits communication between these designer roles. The gap is manifest in the lack of, first, an *integrated holistic methodology* for the complete system life cycle, and, second, *interactive tools* for supporting design activities of this broader life cycle.

First, consider the methodology problem. Despite existing methods for some interface design steps, such as task analysis and user modeling, and guidelines for designing individual displays, there is no methodology which organizes all these steps and guides the designer in assembling the complex structure of an interface. The gap is further manifest by the lack of connection of these interface development methods to methods for software development. Wasserman's User Software Engineering (USE) methodology [Wasserman, 1981] and our own Dialogue Management System (DMS) methodology [Yunten and Hartson, 1985] come the closest to providing the connection. Our research experience indicates that interface development and software development have different kinds of life cycles (see Section 3.3) and correspondingly different methodological needs. These life cycles can be integrated to form a single comprehensive approach to whole system development, but our studies suggest that a unified holistic methodology to embrace activities of both designer roles cannot be achieved simply by extending existing software engineering methodologies.

Second, consider the tool problem. First generation UIMS provide an example of how the few existing tools fail to support activities of the whole life cycle. Early UIMS emphasized run time support over design time tools, emerging from a computer science perspective in which dialogue was programmed, with little initial consideration for supporting the needs of a non-programming interface developer in a natural, effective way. As with the methodology, most UIMS (exceptions again include USE and DMS) do not make any connection with the software engineering life cycle. UIMS support interface development activities, but only during later phases of the interface life cycle, including design specification† prototyping, implementation, execution, and modification. We know of no UIMS that supports the early phases such as task analysis and requirement speci-

† Design specification, as used here, is the process of capturing a representation of a conceptual design (see Figure 2).

fication, and few that effectively support early, creative interface design activities — that tight mental loop of synthesis and analysis involving dialogue scenarios†, experimentation with ideas, "creative doodling," and visualization of results. Without this support, the interfaces produced with UIMS will not necessarily be better than those produced without UIMS. In fact, first generation UIMS may serve only as a means for faster production of bad interfaces.

3. TOWARD SECOND GENERATION UIMS

3.1 Bridging the Gap

Although behavioral scientists and computer scientists have begun reaching toward each other, a gap still exists between their worlds. The move toward second generation UIMS must bridge this gap by developing an integrated holistic approach involving a more extensive life cycle concept, with corresponding tools, to support activities of both designer roles. In an effort to begin bridging this gap, we conducted three studies, the purpose of which was to investigate natural and effective ways in which quality interfaces are developed by human interface designers. By "natural" we mean without *a priori* bias from specific approaches, notations, and tools. Our experiment was not intended to study the cognitive processes of interface designers or of end-users. However, our observations indirectly captured information about their cognitive processes by capturing their observable representation techniques. The key to these studies was *representation* — specification techniques and procedures that interface designers naturally use to capture products of early life cycle activities (i.e., requirements specification, task analysis, conceptual interface design). Based on results of these initial studies, which concentrated on interface design activity, we have concluded that interface development does not "naturally" follow the top-down, step-wise refinement paradigm of traditional software engineering.

† One exception is ACT/1, more a prototyper than a UIMS [Mason and Carey, 1981].

3.2 Observational Data Gathering

The three informal studies† involved a small number of interface designer subjects developing different interactive systems. In the most extensive of these, three subjects developed (including full implementation) a document retrieval system over a period of a year. The subjects were selected outstanding computer science students who performed this as project work for course credit. They were given a written functional description of application system semantics, with no information about the interface and no specific methodological instruction about how to approach the interface design. The three subjects used similar approaches, a composite of which is described here. Their development started with bottom-up, detailed pencil and paper scenarios of what the end-user does, sees, and hears (e.g., pressing keys, entering commands, using a mouse, viewing displays). They initially represented the design with a set of numbered sketches of screen displays, pencilling in beside each item that corresponded to an end-user input (e.g., menu choice, PF-key definition, prompt for a keyword) the number of the corresponding successor screen sketch. They developed a large, flat state diagram to show transitions among display sketches. The result was detailed and concrete, yet large and complex. This early paper and pencil prototype was tested manually and many important changes were immediately fed back to requirements specifications.

Next, moving upward, the subjects reduced complexity by abstracting displays into related groups, forming intermediate and higher levels of abstraction. Simple graph properties were used on the state diagram to reduce or expand detail in subgraphs. The subjects then moved back downward through the representation, analyzing, organizing, structuring, and modifying the design. It was also at this point that the subjects faced issues such as hierarchical language structuring, consistency of keywords and naming, handling

† We do not wish to give the impression that these studies were controlled, m x n factorial experiments with statistical results. Rather, they were protocol gathering observations with qualitative results.

of modes (loops back to non-initial dialogue states), consistent definitions for "quit" and "exit" commands, and structural consistency [Reisner, 1981] of the interaction language. Our subjects had no model or mechanism for organizing and structuring these interface problems. We observed that the DMS dialogue transaction model [Hix and Hartson, 1986] would be especially helpful at this point because of its ability to guide design downward through levels of abstraction, analyzing and refining the interface's linguistic structure.

A first version was implemented, and the next step of development was to alternate back to a bottom-up step of testing with end-users to iteratively refine the interface. Observations here focused primarily on details of the interface and how the user perceives them at the lowest level of abstraction. Here many of the detailed design decisions (e.g., screen layout, menu format and content, graphics, consistency of wording) were made and/or changed, ones that could not be addressed at higher levels of abstraction. Before end-users (including us) were reasonably satisfied with the quality and functionality, the designer subjects had gone through three complete revision cycles, some affecting the deep structure, and many iterations for fine-tuning interface details.

We conducted two other less extensive, but nonetheless informative development studies. One involved a database of student information for corporate recruiters developed by a group of 15 computer science seniors (also implemented and tested with users); the other involved a small accounting system developed by four computer science graduate students (not implemented). All three of our subject groups arrived independently at some form of scenarios and state diagrams as a starting point for development.

Two other independent occurrences give added credence to our observations. First, for the last year and a half, we have been working with IBM Federal Systems Division on extending their approach to interface development. Delving into their current approach, we found that much of their development work is scenario-driven. Often, the customer provides these scenarios; on other occasions, the interface developers produce them as the first step in the development process. Second, Carey [1987] relates observations of numerous independent design teams; all produced paper and pencil versions of the screen sequences early in the development process.

3.3 Result: A Dialogue Development Life Cycle

It is probably impossible to determine empirically (or otherwise) the "best" way to develop interfaces. But our research yielded some indications of what some interface developers want for representation, specification, design, and implementation techniques and tools to support their own personalized approaches to the early, creative phases of interface development. Based on our observations, we hypothesize that the typical initial approach is bottom-up, based on "story-board" dialogue scenarios and often augmented with state-diagram-like representations of sequencing. There are later parts of the development process, however, that benefit from a process of top-down, step-wise decomposition and structuring. Despite claims to the contrary [Draper and Norman, 1985], our observations have led us to hypothesize that the life cycle activity for interface development is necessarily different from that of the traditional software development life cycle, with its top-down linear sequence of requirements, design, implementation, and testing†; and that the overall system development life cycle is actually made up of alternating "waves" of what we have come to call "Type A" and "Type B" activities, summarized in Figure 1. Type A activities are top-down, analytic, and structuring. They involve modeling and reflect a system view. Type B activities are bottom-up, empirical, synthetic, and reflect the end-user's view.

Although task analysis and requirements specification were not the focus of our obser-

† However, others [Swartout and Balzer, 1982; Ramamoorthy, Prakish, Tsai, and Usuda, 1984] say that this linear cycle is not the most appropriate for software either. Given that view of software engineering, the arguments of [Draper and Norman, 1985] for treating interface design analogously to software engineering are more compelling.

Type A	Type B
Top-down	Bottom-up
Analytic	Synthetic
"Stop"	"Go"
Organizing	Free-thinking
Judicial	Creative
Structural	Behavioral
General	Detailed
Modeling, formal	*Ad hoc* or empirical
Reflect system view and works toward user	Reflect user's view and works toward system
Language orientation	Dialogue flow orientation

Figure 1. *Summary of Type A and Type B Activities*

vations, it appears that our theory of alternating up and down waves continues to apply. Task analysis is top-down and analytic. It starts with the whole system and breaks it down, often hierarchically, into functions, tasks, and subtasks. Also probing earlier in time, we see that task analysis could be driven by a bottom-up process of early and incomplete requirements specification.

To summarize, the development process seems to be comprised (although not in a strict linear time sequence) of:

Bottom-up:	early incomplete requirements specifications
Top-down:	hierarchical task analysis
Bottom-up:	scenario-driven interface design
Top-down:	interaction-language-driven abstraction and structuring
Bottom-up:	empirical user tests to improve interface
Top-down:	modeling and abstraction to reorganize and restructure
⋮	⋮

We have extrapolated our observations into a tentative model of the interface/system development life cycle, shown in Figure 2. This is the kind of life cycle we foresee being supported by second generation UIMS. There are obviously some similarities between this figure and other system life cycles. However, only a few other researchers include interface design (e.g., [Wasserman, Pircher, Shewmake, and Kersten, 1986]). The life cycle in Figure

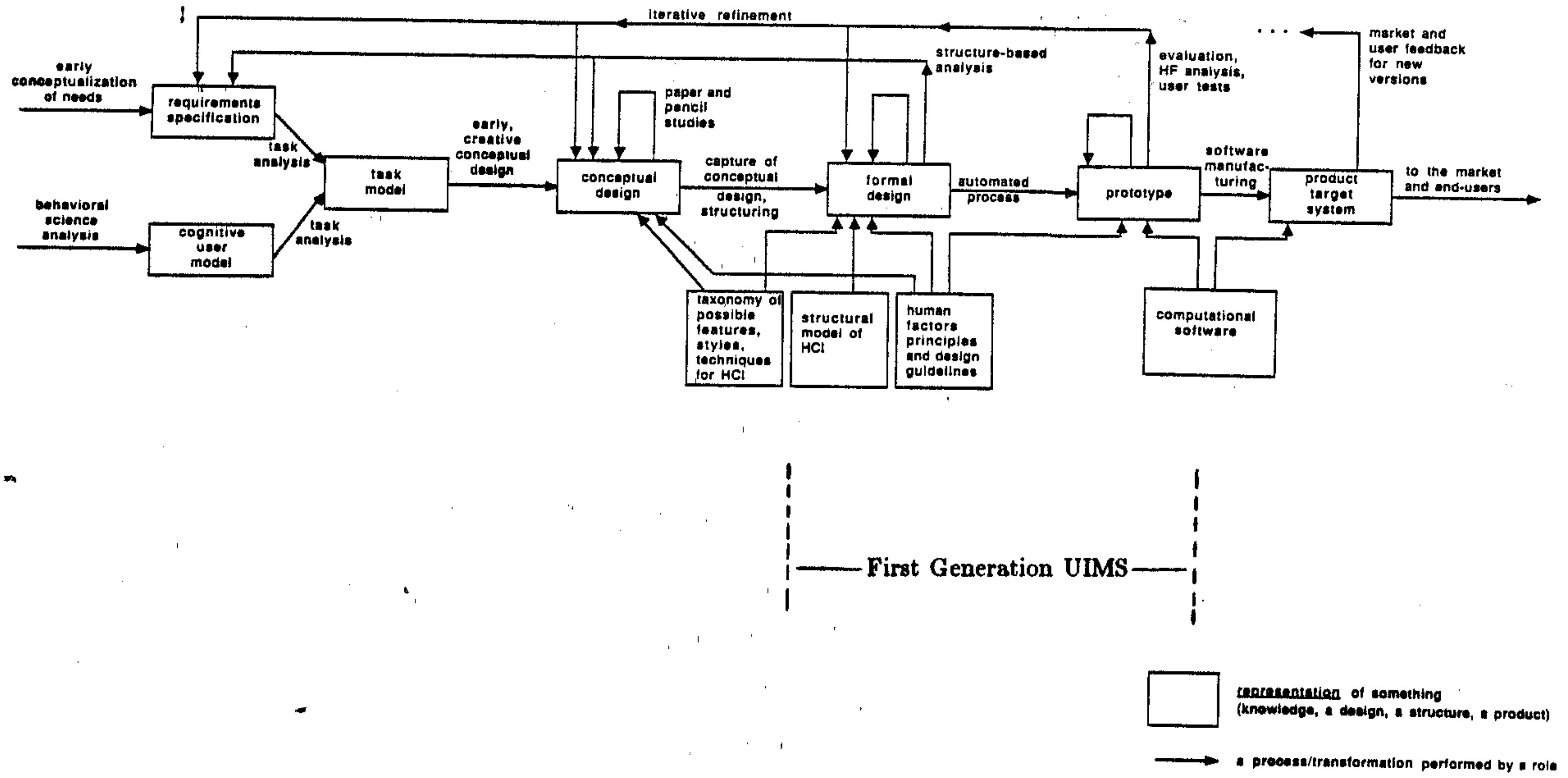
iterative refinement
structure-based analysis
early conceptualization of needs
requirements specification
task analysis
behavioral science analysis
cognitive user model
task analysis
task model
early, creative conceptual design
conceptual design
paper and pencil studies
capture of conceptual design, structuring
formal design
automated process
prototype
evaluation, HF analysis, user tests
software manufac-turing
product target system
to the market and end-users
market and user feedback for new versions
. . .
taxonomy of possible features, styles, techniques for HCI
structural model of HCI
human factors principles and design guidelines
computational software
First Generation UIMS
representation of something (knowledge, a design, a structure, a product)
a process/transformation performed by a role

Figure 2. *An Interface Development Life Cycle.*

2 focuses more on the early conceptual phases and has more feedback loops for iteration. Because of the highly cyclic nature of the diagram, almost any place is appropriate to enter this life cycle. Thus, a later entry point into the cycle of Figure 2 (e.g., "playing with scenarios," a conceptual design activity) may be followed by feedback to earlier points (e.g., formulation of requirements specifications) in the life cycle. Such an occurrence runs counter to traditional, linear top-down paradigms, but supports our hypothesis of alternating waves of development.

In Figure 2, specification of requirements is a functional description of application semantics of the whole system and serves as a problem statement for the development process. Early specifications are usually quite incomplete and rarely reflect the functionality of the final product. The cognitive user model explains how the user thinks about tasks performed with the application system. Task analysis produces a model of user and system activities and behavior, based on expected system functionality. Early design work involves some of the most creative activity and rapidly changing results. The representational products of this early design work (e.g., the conceptual design) might include scribbles on an envelope, a mental image in the designer's mind, or a written sequence of screen scenarios. The conceptual design is often incomplete, linear, unstructured, and not computer-stored. The design process now increases its pace, as a first generation UIMS enters the life cycle. The design is now captured in computer form; it is refined, alternatives weighed, and details added. Major changes often occur from conceptual design, and, depending on UIMS functionality, considerable structuring is added. Representation of the formal design can take many forms [Hartson and Hix, 1987], such as BNF, state transition diagrams, hierarchical interaction languages, and WYSIWYG screen layouts. Many UIMS can, at this point, produce an executable prototype, for evaluation and iterative refinement. Some can add realism by merging the functionality of any implemented computational software into the prototype.

3.4 The Importance of Representation

The most important key to bridging the gap between behavioral scientists and computer scientists in interface development is *communication.* The key to this communication is *representation* — a clear, unambiguous, easy-to-visualize mechanism to convey the specification and design of the evolving interface to both roles. Our observations showed that the two different designer roles need a common representation for communicating but have different needs for representation in their own separate design and development work. A solution lies in the proper kinds of *abstraction,* one that allows a common view for sharing, but one to which *filters* can be applied to produce different working views for each role. It is difficult to maintain multiple views with manual (e.g., paper and pencil) means, but is achievable with computer-based tools such as UIMS. A single representation is maintained internally from which various views are derived on demand.

In software engineering, textual specifications, design documentation, and implementation code are all various kinds of representations of the target system. Production of such representations is a physical task, regardless of whether interactive tools or manual methods (e.g., pencil and paper) are used. A corresponding mental task to create the specification, design, or implementation necessarily precedes and accompanies the physical task of representing it. This sequence of mental and physical activity is illustrated by the boxes labelled "conceptual design" and "formal design" in Figure 2. With first generation UIMS, the mental and physical tasks tend to be quite separate, causing discontinuity in the interface developer's work. The second generation of UIMS should bring the mental and physical tasks closer together by providing effective interactive tools that closely match the mental processes they physically support.

4. FUTURE DIRECTIONS

Existing limited representational techniques (e.g., Backus-Naur Form grammars, state transition diagrams) and related tools were brought into the interface development process primarily by computer scientists. To our knowledge, however, there is no empirical support for their efficacy as a natural or a complete representational technique for a interface developer in a real design environment. Our work reported here, although preliminary, is some of the first in which experimental studies have been used to drive the development of support methodology and tools. It is our goal that the interface (and whole system) development process will become a smooth, continuous, tool-supported process almost from the beginning, using a natural means of representation that makes it easy to capture the design, easy to experiment with it, easy to visual interface behavior, and easy to change it.

Within our DMS project we are continuing the initial work presented in this paper. This includes extensive protocol gathering of expert interface designers at work on bench-mark interface tasks, with special emphasis on their task analysis, requirements specifications, and design representation. Analysis of these observations will increase our understanding of representational needs in these early activities and, we hope, allow us to derive corresponding methodological and tool support. It is our hope that this will help bridge the gap between the creative design process and the implementation process (e.g., UIMS and software engineering), taking us to the next generation of UIMS and, eventually, beyond.

ACKNOWLEDGMENT

We wish to acknowledge the participation in our DMS research by our colleague and friend, Dr. Roger W. Ehrich. We also wish to thank Pat Cooper for her speedy typing, without which we would never have met the deadline. The DMS research is funded by the National Science Foundation, Dr. H. E. Bramford, Jr., Program Director; IBM Federal Systems Division; 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, 35-42.

Card, S. 1986. Opening remarks at *CHI '86 Conference*. Boston. (April).

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

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

Flanagan, D., Lenorovitz, D., Stanke, E., and Stocker, F. 1985. RIPL Concept of Operations and System Architecture. Internal document. Computer Technology Associates, Inc., Englewood, Col. (May 1).

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

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.

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.

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.

Hix, D., and Hartson, H.R. 1986. An Interactive Environment for Dialogue Development: Its Design, Use and Evaluation. In *Proceedings of CHI '86 Conference.* Boston. (April), 228-234.

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

Mantei, Marilyn. 1986. (To be obtained.)

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.

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.

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

Reisner, P. 1981. Formal Grammar and Human Factors Design of an Interactive Graphics System. *IEEE Transactions on Software Engineering, SE-7* 2 (March), 229-240.

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.

Sibert, J.L., Hurley, W.D., and Bleser, T.W. 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., 1987.

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.S. 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., 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.