An Agenda for Human-Computer Interaction Research: User Interface Development Processes and Methodologies

By H. Rex Hartson

TR 91-17

AN AGENDA FOR HUMAN-COMPUTER INTERACTION RESEARCH:

USER INTERFACE DEVELOPMENT PROCESSES AND METHODOLOGIES

H. Rex Hartson
Department of Computer Science
Virginia Tech, Blacksburg VA

This paper is the result of one working group in a workshop entitled "An Agenda for Human-Computer Interaction Research: Science and Engineering Serving Human Needs." The workshop, sponsored by the National Science Foundation, brought together "20-25 of the most prominent HCI researchers from the disciplines of computer science, engineering, information science, psychology, and human factors along with several NSF staff members." The workshop results, to appear in the HCI literature, identify critical research issues and potential avenues for assaulting them, along with necessary infrastructure recommendations related to educational, professional, and facility problems. The overall topical area was divided into five areas, each with an individual researcher in charge of directing discussion and reporting on the area. The areas and researchers in charge are:

Theory and taxonomy of HCI models	Stuart Card
The interface development process	Rex Hartson
I/O devices and interaction styles	Rob Jacob
Software tools	Dan Olsen
Computer supported collaborative work	Judy Reitman-Olsen

This present paper is the report by the group chaired by the author on the topic of the interface development process.

AN AGENDA FOR HUMAN-COMPUTER INTERACTION RESEARCH:

USER INTERFACE DEVELOPMENT PROCESSES AND METHODOLOGIES

H. Rex Hartson
Department of Computer Science
Virginia Tech, Blacksburg VA

1. INTRODUCTION

1.1 Background

The goal of many people working in the human-computer interaction (HCI) area is better interactive software systems. How can HCI contribute to that goal? One way is to produce systems with better interfaces. It is obvious that HCI research will aid in this by offering ways to produce interfaces with increased usability.

Another way is to develop new kinds of systems to support particular kinds of work (e.g., group work). Here the nature of the application is the direct subject of HCI research. A third way is based on the premise that better software systems are influenced by their development process. The development process is supported by tools, and there is also an HCI role in improving the tools (especially their usability for their users, the interface developers). In this paper we shall focus on the user interface development process, and a research agenda for progress in this area.

Most of these research issues come under the general heading of methodologies that prescribe and structure activities within the development process. By methodologies, we mean theory-, principle-, and model-based methods and techniques that have been validated to determine their true effectiveness.

There is a need for both "big" and "little" research on HCI development processes. Big research will bring about new theories and major breakthroughs that change the way we think about the problems. Little research is also needed to find solutions, often low technology solutions, to pragmatic problems—essential to the transfer of technology from the laboratory to the workplace. Finally, before proceeding, because some of the terms used here have a similarity to terms in software engineering, it is important to clarify that this paper is about HCI and user interface development and not about development of software.

1.2 Justification

While development methodologies cut somewhat across the other areas, especially across software tools to support interface development, they also constitute an important research area in their own right. Methodology is often the vehicle for a theory-based discipline to translate theory into practice. In the original announcement of this workshop, the first paragraph for "An Agenda for Human-Computer Interaction Research: Science and Engineering Serving Human Needs" ends by saying (emphasis my own): "The interactive systems program [of NSF] has been aimed at supporting the discovery of basic principles of human computer interaction and the development of innovative software tools and *methodologies.*"

Much of the current research attention seems to focus on HCI theory or tools or product-oriented ideas, with less concern for methodology. Development of user interfaces is the process where the discipline is put into practice. Methodology draws together theories, models, approaches, and processes and organizes them to act in concert to produce a highly usable and functional product. Today most of these processes are still ad hoc, resulting in quality that varies with the experience of the developers. Thus, there is a need for HCI research in this area, to put these processes on a solid scientific footing.

1.3 A Framework for Research in User Interface Development Methods

The various activities that comprise the user interface development process lend themselves as topical headings in an outline for organizing the subject. However, the danger in using them as such is the possibility of the misconception that these activities can be cleanly separated in practice. Such a clean separation is the premise of the phase-oriented "waterfall" software development methodology, which we know does not work for development of user interfaces. Some contend that it likewise does not even well serve software engineering (Boehm, 1988).

On the other hand, although these development activities are intertwined in an iterative cycle, the activities themselves are usually distinguishable and so do provide a useful framework for structuring our discussion. This framework serves well as a context for discussing and comparing various development methodologies. In particular, few methodologies, if any, apply to the entire spectrum of development activities. This framework provides a yardstick for determining and comparing the coverage.

1.4 Organization of this Report

The sections of this report are, thus, organized around individual interface development activities. In one sense considerable progress has been made in many of these areas. But in another real sense, we have exposed only enough to realize the need for real breakthroughs. Within each section there are two subsections: an introductory subsection and a subsection to highlight key research issues concerning that activity. The introduction describes and defines the activity and motivates its importance to human-computer interaction. This will often include a brief review of the current state of the art, including references to related work in the literature, where appropriate.

This report is not a survey, however. It is assumed that the audience is the HCI research community, plus others who will benefit from being informed about current HCI research directions and the importance of HCI in computer science. Therefore, the review of related work largely depends on a few representative references to existing work to serve as pointers for readers in need of more detailed discussion, and no attempt is made at completeness, as there might be in a survey.

The subsection describing key research issues provides an explicit list of research goals and directions for progress in the future, the primary purpose of this report.

1.5 Approach

It is fairly easy to distinguish, in user interface discussions, between product and process. The *product* is the resulting interface design (and its implementation), including objects, their behavior, semantics, content, style, and human factors of a given interface. What is reported here is not about the product and, therefore, rules out most discussion about such things as principles and guidelines for interface content (e.g., calling for consistency), except when such guidelines might affect the process itself. In this paper, rather, we focus on the *process* by which these things are developed, integrated, and tested within an interface design.

For purposes of discussion, we should also like to distinguish between the terms design and development. In this context the term "design" (used here as a verb referring to a development activity) is used to apply (somewhat narrowly) to the creative human mental problem-solving activity used to synthesize tasks, objects, and features and put them together to make up a new user interface design (used here as a noun to refer to a product). We use the term "development" here as a broader verb, encompassing design along with systems analysis, needs analysis, user analysis, requirements analysis, task analysis, functional analysis, task allocation, design representation/specification, rapid prototyping, evaluation, design of an implementation model, programming, software "manufacturing" (polishing/optimization), deployment, documentation, maintenance, and iterative refinement.

The term development can apply in different ways to many parts of an interactive system. Figure 1 shows one view of relationships among various kinds of interface development within overall system development. This diagram is intended as a means for organizing a discussion of the topic area and not an architectural diagram of an interactive system or user interface management system.

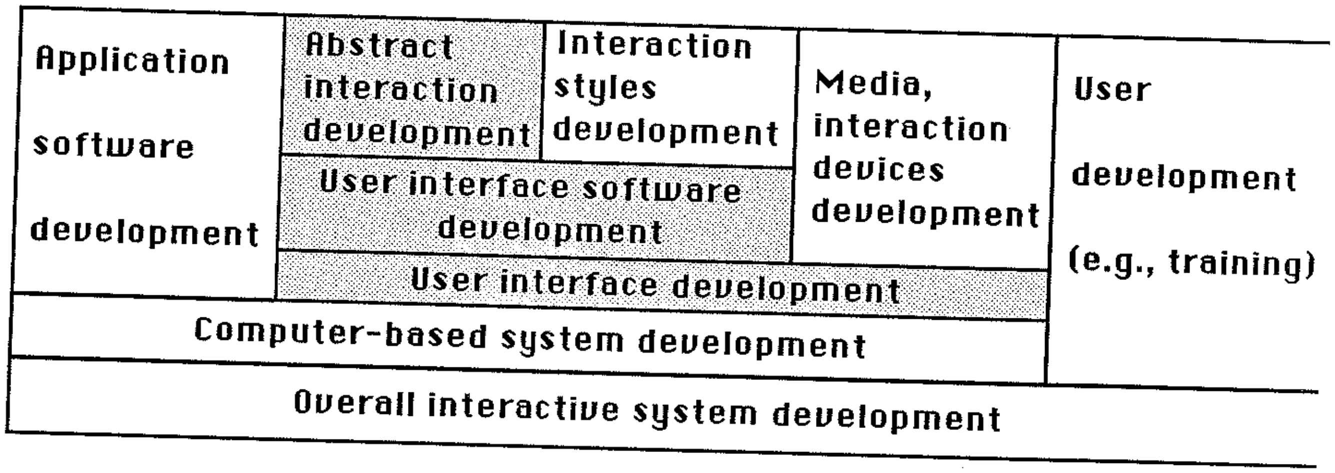
Application software development
Abstract interaction development
Interaction styles development
Media, interaction devices development
User development (e.g., training)
User interface software development
User interface development
Computer-based system development
Overall interactive system development

Figure 1. Relationships among interactive system development activities

Figure 1 shows that user interface development is part of a larger set of development processes. From the outside in, we have overall interactive system development, development of the part of the interactive system that is computer-based, user interface development, user interface software development, and abstract interaction development. Each of these domains of development involves its own lifecycle with its own set of processes. Except for projects to develop new computing systems or workstations, user interface developers usually find that choices of hardware and interaction devices are made independently. In this paper we are interested in the highlighted areas of Figure 1: primarily the domain of abstract interaction development, in the context of user interface software development, and user interface development in general.

Some of the development activities mentioned just before Figure 1 apply solely to development of abstract interaction (e.g., task and user analysis), some solely to development of interface software (e.g., programming), and some apply to both (e.g., design and representation). To set the stage, we see user interface development depending on two tenets in tension: on one hand user interaction design is completely separate from software design; on the other hand the interface *is* an integral part of an interactive software system.

As shown in Figure 1, design of interaction with the user has two separate and essentially different embodiments: the abstract interaction component and the concrete software that implements the interaction. Design and development of the abstract interaction are different from design and development of user interface software. The two kinds of activities require very different skills, attitudes, perspectives, techniques, and tools. Software is the implementation medium. In principle, this is only an "accident" and the medium could have been, say, clay or Legos. The point is that it is useful to be able to discuss user interfaces independently of implementation, software, and programming. Design of the interaction becomes part of the specifications or requirements for design of the interface software. Software itself also breaks down (at least conceptually) into interface software and non-interface (computational) software, although that separation, as we know, can be elusive and difficult to define at times.

The separation between user interaction and interface software must be balanced against the fact that the user interface is part of an interactive system, and its development ultimately must be treated as an integral part of the overall system development process. To this end, some have proposed a parallel between the interface development and software methods—e.g., (Draper & Norman, 1985).

2. ITERATIVE METHODOLOGY

2.1 Introduction

Although it is not always clear to developers, user interface development must *necessarily* be iterative (Gould, Boies, & Lewis, 1991; Gould & Lewis, 1985; Hartson & Hix, 1989). Unlike software engineering, where development can be top-down and correctness-driven, user interface development must be self-correcting. That is, interface development must be a form of trial and error activity, its correction depending on feedback from evaluation. Iteration is employed in both software engineering and user interface development because of our inability to account for all the details at once. However, iteration is used in user interface development for an additional reason. Although we do have some methods for predicting software behavior, we cannot predict human user behavior. And this need for iteration isn't just until we can know enough to get it right the first time, " . . . but because in a design domain we can never know enough" (Carroll & Rosson, 1985). However, the software engineering of user interface software can be a top-down part of the iterative user interface development process, as shown in Figure 2.

Here software engineering of the user interface software can be top-down in the sense of matching the user interface software to the interface software requirements as produced in each iteration of the abstract interaction design.

As with most creative processes there is a need for an alternation of synthesis and analysis (Hartson, et al., 1989). This is probably true, in reality, for software engineering as well. In some of our empirical observations of real world software engineers at work we noticed iterative and alternating development activities but, because corporate standards and methods required it, work was reported as having been done strictly top-down. As a result of these observations, we proposed an iterative, evaluation-centered "star" lifecycle instead of the linear "waterfall" process of moving from one phase to another. Many times software engineering activities are from the "inside out" with the major work being the development of a good "impedance" match between what is at hand at the bottom and what is expected at the top. To be effective, the developer must acquire a comprehensive understanding of the connections of these elements at all these levels. Understanding the development process requires an understanding of how developers acquire this comprehensive overview. The same appears to hold in HCI. To some extent, the same probably holds in the development of any large design (e.g., the complete design of a large building, the design of a large modern airplane).

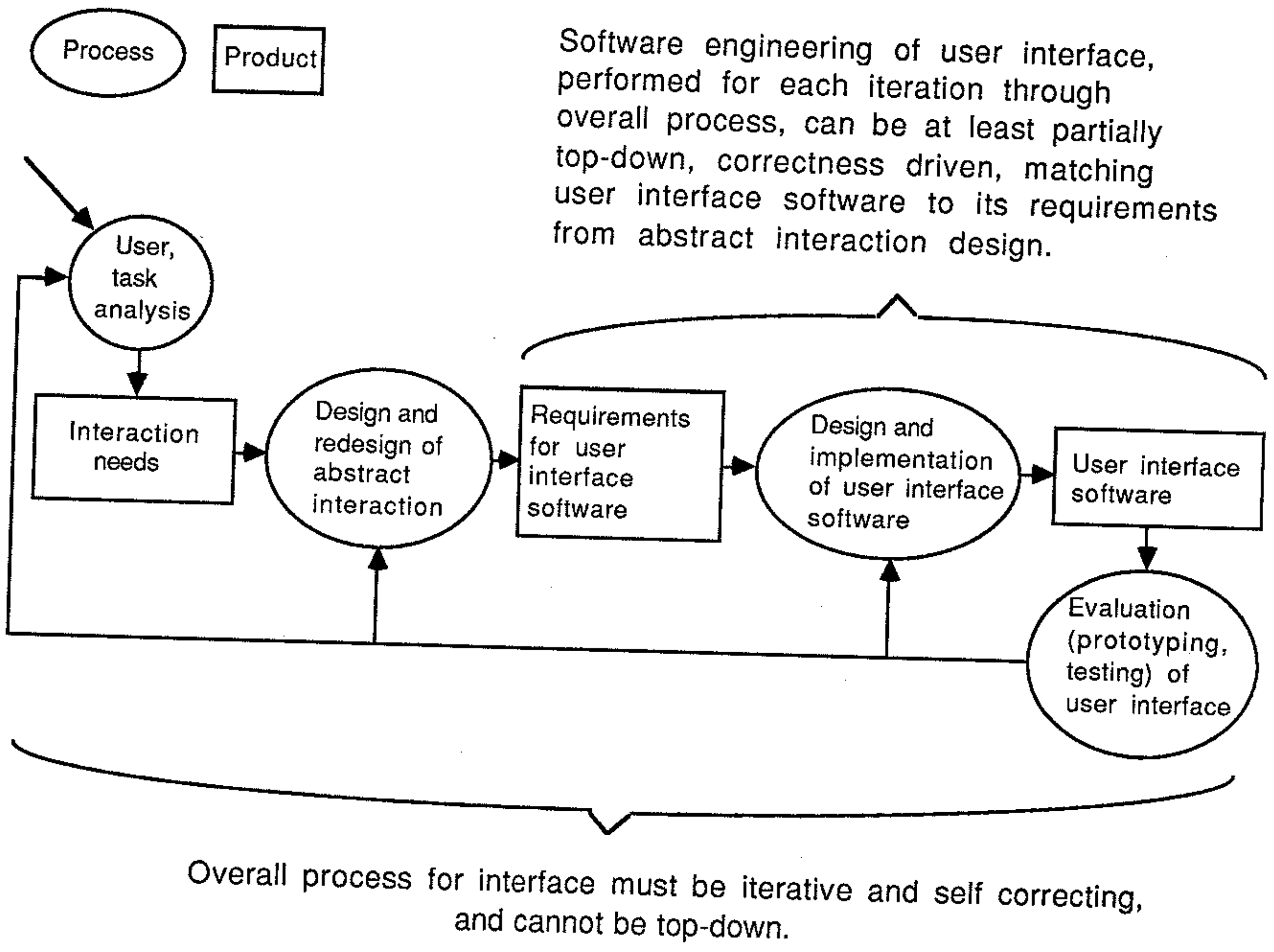
Process
Product
Software engineering of user interface, performed for each iteration through overall process, can be at least partially top-down, correctness driven, matching user interface software to its requirements from abstract interaction design.
User, task analysis
Interaction needs
Design and redesign of abstract interaction
Requirements for user interface software
Design and implementation of user interface software
User interface software
Evaluation (prototyping, testing) of user interface
Overall process for interface must be iterative and self correcting, and cannot be top-down.

Figure 2. Iterative user interface development process, containing software engineering of interface software

Beyond these similarities, however, there are fundamental underlying differences between iteration for software engineering and iteration for user interface development. First, the most significant purpose for iteration is very different, and that difference in purpose leads to the need for completely different methods for carrying out and controlling the iteration. A goal in software engineering is to achieve correct and complete functionality. Iteration of a software design is used to ensure that requirements are eventually correct and that data structures and functional algorithms are correct and complete implementations fulfilling the requirements. The kinds of iterative activities that apply include formalization of requirements and axiomatic validation of implementation to prove that design and coding of the programs and data structures meet the requirements. Some iterations might be more bottom-up to design data structures and put together details. Other iterations might be more top-down in terms of structuring modules and routines into bigger programs, sharing and reuse of routines, etc. Formative evaluation is done to ensure the functionality is as complete as possible and that the software is working correctly.

A goal in developing the user interface is to achieve high usability in the way the system interacts with the user, including high learnability, high user task performance, low error rates, and high user satisfaction. Here task-based usability specifications are quantifiable inputs to the abstract interaction design, stated in terms of human user performance goals. In contrast, software engineering outside the domain of the user interface is not concerned with usability specifications. Further, for user interface development, testing is with human users, measuring user performance and satisfaction; this kind of testing is not appropriate for software engineering (outside the interface domain). Development of the interface component is deeply concerned with the psychology of the user; software engineering is not.

This profound difference in the way development in each of the two domains uses iteration is related to the first tenet of the two "tenets in tension" discussed in the introductory section, that development of the abstract interaction part is separate from software engineering.

2.2 Key Research Issues

A research agenda associated with the iterative approach includes finding solutions to a combination of technical, management, and technology transfer problems. Specifically, these problems include getting usability out of the laboratory and on the front lines and finding ways to control the iteration (i.e., knowing when to stop). New techniques are also needed to ensure that the process is converging and not *just* iterating. And, of course, there are difficulties with communication among the multiplicity of separate, but cooperating, developer roles. Solutions to the problem of convergence toward an improved design are tied to new developments in formative evaluation, and this topic is discussed in more detail in the section on evaluation.

Iterative feedback from evaluation to design must to be grounded in a science base that connects features of a situated artifact (i.e., in a situation of use) to psychological consequences for the user (Carroll, 1990; Carroll & Kellogg, 1989). To illustrate, a system that offers examples for learning is making certain claims about how people learn and what activities they prefer. The nature of the examples makes certain claims about what activities are typical in this domain. The design can also be grounded more broadly in a psychological science base, which addresses the role of examples in learning. This connection of evaluation activities to principles can be used to help organize redesign effort. A developer can ask questions regarding what is good about a new version of an artifact, and what claims, which when analyzed, are true and important in determining users' experience and yet were untrue in earlier versions. Since artifacts can be typed and abstracted, this analysis can also help cumulate further science, allowing analysis for one artifact to be sharable with others.

3. SYSTEMS ANALYSIS

3.1 Introduction

Systems analysis refers here to activities such as needs analysis, user analysis, requirements analysis, task analysis, functional analysis, and task allocation. Just like everything else in HCI, these activities, too, are iterative and interwoven with later activities such as design. It is obvious that careful attention in these early stages of interface development should lead to easier, faster, and more economical achievement of quality in the final product. Nonetheless, most developers want to get past the early activities as fast as possible and get on to the better understood parts of the process such as design and evaluation.

Task analysis is an especially important component of this early analysis activity. Task analysis appears to be more well developed and more extensively practiced in Britain than here in the U.S. Yet there are important reasons for approaching design from a user task orientation. This is because the task domain is where the designer determines what kinds of users a system will have, what the users wish to do with the system, and how the users will go about performing functions with the system. User-centered design must be driven with methods and approaches that account for cognitive needs and limitations of the user in the context of tasks. (See the discussion of behavioral vs. constructional domains in Section 5.1.)

3.2 Key Research Issues

A great deal can be done to model and formalize systems analysis activities, such as needs analysis and user analysis, and to include them explicitly in development methodologies. Much of the bad design that must be filtered out in the iterative process can be avoided early on, if the real goals of the design and the characteristics of its intended users are more thoroughly considered. This is an area of very little current HCI research, an area open and ripe for more theory and codification of techniques. For example, there is a need for observational empiricism up-front as an input to requirements definition (in contrast to its more obvious use in the later evaluation activity).

The early processes can have an enormous impact on the final product, and on the ease and cost of reaching it. Yet these processes are among the least well-defined and least formalized and are not well understood or extensively practiced by the majority of user interface developers. Specific areas for research include field studies of real development projects, essential in defining these activities and the associated developers' roles. These studies can be used to determine, through observation and deduction, an operational model of what real interface developers presently do, as well as what they should be doing. Only through field studies of real interface developers can we hope to understand the day-to-day operational requirements and constraints placed on developers and the needs they have for methods and tools.

Of all of the systems analysis activities, task analysis is one which may benefit most from having a specific research agenda. Task analysis has been poorly adapted into HCI from human factors. Task analysis in HCI has gotten lip service, and it has been used occasionally by some practitioners (mostly those who knew how to do it in other settings). But a specific prescriptive methodology for doing task analysis in user interface development is lacking.

Enormous operation sequence diagrams are still used to give great detail about specific paths through a task. But, because these are sample execution traces (and, therefore, extensional), they are not easy to bring to bear on the general design problem in HCI. Because HCI design has such a large domain of asynchronous task possibilities, it requires an approach that is essentially intensional—i.e., addresses all things that *can* happen. Also, operation sequence diagrams typically represent information such as "here the user makes a decision and, based on that decision, selects a menu choice." But at that point, if designers understand what information is needed to make the decision and what information is available, they would be in a much better position to support what the user really wants to do in that task.

There is a strong research mandate to find new ways to get a deeper understanding of the user's tasks into the interface design, bringing task analysis fully into the interface development process. Most task analysis for interface development is done now mainly in terms of physical actions a user makes in carrying out a task. An interface designer can do a much better job of supporting the user in the performance of a task if the design is based on a task analysis that also includes more of the cognitive, memory, perceptual, and decision making actions. Specific research is needed to find ways to model how information is acquired and utilized by users while performing a task.

A valuable place to begin is a comprehensive review of task analysis techniques, adapting, recasting, extending, and modifying them into a form which would a priori be matched to the needs of the HCI design/development process. What is learned can be used to build a model (and a methodology based on the model) for task analysis within the interface development process. This must then be validated by an investigation of whether the result really is matched to the needs of the design/development process.

There are, of course, some user modeling techniques that offer the needed kinds of task operators. The GOMS model (Card, Moran, & Newell, 1983) and the Command Language Grammar (Moran, 1981) provide an important basis for task analysis in HCI. However, they need adaptation into the development process, because in this role their purpose is less to support analysis to predict user performance and more as a *descriptive* representation of the task from a design/synthesis perspective. In the design environment, our understanding of tasks is shallow. Here we need to know more about the users' needs (cognitive, perceptual, information processing, etc.) during the performance of a task and whether these needs are supported in the design. Higher levels of abstraction are needed so that less detailed task representations can be formulated more quickly than allowed by the keystroke level of detail (Card & Moran, 1980) normally associated with these performance prediction tools. Also more emphasis needs to be placed on supporting the user in the task. For example, Fitts' Law can be used to analyze mouse movement in a design or the keystroke level model to analyze that kind of physical action, but the most difficult part in understanding the user and the task may be in questions about *why* the user is moving the mouse or making the keystrokes.

As part of the task analysis research agenda, there is a need for research on task taxonomies within HCI. Results can help with reuse of task analysis in connection with a new task. At the present we do not have a taxonomy to help us understand how tasks are related. The act of taxonomizing in itself, by organizing and structuring our rapidly growing HCI knowledge base, would be of great benefit. Taxonomies of tasks, user classes, methodologies, interface objects, interaction styles—among others—would be very useful to designers. Also matrices showing relationships (e.g., tasks by methodologies) would have significant practical value. This latter would include a structured, prescriptive methodological connection from task analysis to design. Having done task analysis, the designer still presently does not have a method for proceeding smoothly and naturally to the design process, the subject of the next section.

4. DESIGN OF THE INTERACTION COMPONENT

4.1 Introduction

Probably as much has been said in the literature about user interface design as about anything in HCI. In the early days of HCI, much of it was *ad hoc* folklore, "principles" for design without empirical basis. Probably the most creative and individualized activity in the development process, design is also the most difficult to understand and formalize. So we must be content to make up for our ignorance through constant evaluation and mid-course corrections. To really understand user interface design is to understand design in general, a topic that has been studied for many decades without yielding generally applicable formulae for success.

For the sake of clarity, it is important to emphasize that, in this paper, discussion of the design activity is about design of the interaction part of the interface and not design of the interface software. Design, even as we have narrowly defined it here (see Section 1.5 on Approach), is a major activity in the development process; it is where the substance and appearance of the interface are synthesized. Because of its intimate relationship to other development activities, most work in this area goes beyond pure synthesis and often includes iterative and evaluation-related aspects. The work by Carey (1982) and Rosson, Maas, and Kellogg (1987; 1988) are representative of the broad work in this area. In addition, other papers appeared in the CHI '90 proceedings—on the task-artifact cycle (Carroll, 1990), contextual design (Wixon, Holtzblatt, & Knox, 1990), and use of "critics" (Fischer, Lemke, Mastaglio, & Morch, 1990). We need to bring cognitive science to bear more directly on the user interface design process. As Carroll (1990) puts it, a goal of HCI in the 1990s is "to produce applied psychology that efficiently adds value to design work." Much of what has been said here, of course, applies to the whole development process (especially evaluation), but these things are mentioned here because they have a significant impact on design and re-design. These ideas should be noted in the context of this workshop as important new theory- and method-based directions being taken to provide direct techniques for producing high usability designs.

Interface design is required at several levels of abstraction. At the main level of abstraction, most applications are presented to the user as a workspace (holding and displaying a document, spreadsheet, drawing, etc.), plus a collection of the main application functions. These functions may be accessed through pull-down menu choices, selectable buttons, palettes, etc. One can call this the *functional* level of design because of the large number of application functions simultaneously available to the user. There is also a level of abstraction below the functional level, which one could call the *articulatory* level because it contains details about how functions are invoked and applied, parameters supplied, etc.

4.2 Key Research Issues

A key research direction for work to support the design process is faced with unsolved problems regarding how the basic functionality of an interactive system, in the functional level just discussed, is formalized. Research is needed on formal modeling and representation at this functional level. It is possible, for example, to represent an application function formally as a mapping, with inputs and outputs, that transforms the application product (e.g., document) into a new state. Such a representation would support composition of functions into larger functions and would allow formal reasoning about the nature of the application functions. It would also lend a strong foundation for desirable interface characteristics such as consistency. A connection to research on task allocation, which is used, for example, to decide which function inputs are provided by the user and which by the system, would support these essential considerations in the design process.

While most of the design effort for an interface goes into specific artifacts and how the user interacts with them within a task at the functional and articulatory levels of abstraction, there is another important level, possibly above this functional level, which one could call the *user strategy* level (Draper, 1989). If this level exists in an application, it contains a model for helping the user determine strategies for carrying out higher level goals and intentions. This is the level that deals with how to use the application. Yet this level is not included in interface designs for most current applications and, therefore, is a key area requiring research.

Another research goal in the area of interface design is to discover helpful, unobtrusive, structured, and organized ways to integrate the use of principles, guidelines, standards, style guides, and design rules into the design process without stifling creativity. To a designer, the guidelines can be too voluminous (e.g., nine hundred and forty-four from Smith and Mosier (1986)). It is especially difficult to translate a general, often abstract, set of guidelines into specific, concrete design decisions. Perhaps interpretation of the more general guidelines provides designers with creative freedom, within the bounds of HCI principles. We are presently far from knowing how to offer designers access to the right guideline for the right use and how to deal with inconsistencies and vagueness within the guidelines. Research issues include methods for offering the designer assistance in understanding, searching, and applying design principles, guidelines, rules, and standards. This need appears to go well beyond what might be provided by, say, an expert system to suggest design rules during the design process. Another issue is how best to communicate design rules and ensure that they are observed. Modern graphic interface designs may require several thousand rules for complete specification. New tools and methodologies are needed to deal with these issues.

There is also a connection to task analysis. If design guidelines can be classified into a framework that correlates to task characteristics, knowledge that certain characteristics make more difference than others regarding a specific kind of user performance could lead to helpful choices of design guidelines that specifically address usability goals and the design could be tailored accordingly. For example, if error-free performance is most critical, then design guidelines most important for this characteristic can be determined.

Another research directive for the design process is one resulting in shared access to principal-based and empirically tested state-of-the-art design features. While toolkits such as X Windows, Motif, etc. provide shared access to interface objects, design of these objects is not strongly principle-based or empirically tested. More importantly, these toolkit objects are shared within software implementation environments. In addition, there is a need for libraries of artifacts (e.g., scrolling list in a dialogue box) that have been developed, tested, and refined—to be shared within a given task environment, independent of software. This would lead to increased productivity by avoiding reinvention, by raising the level of abstraction in design work, and by providing necessary information to allow artifacts to be deployed successfully within interface designs. The work of Mackinlay, Card, and Robertson (1990) is in this direction, but needs a translation to put it into practice.

The design of help functions involves other research issues to be addressed. Consideration of help for the user must be integrated throughout the overall development process and not just an add-on at the end, as is often done in present practice. At each activity in the development process, starting with early predesign and into design, one needs to think about how the structure of the interface influences the ability of the user to get help. For example, simple task analysis would reveal that it is not useful to remove the currently on-going task to display the help and then erase the help information when returning to the original task. Usability specifications also ought to include use of help and other built-in training functions.

Documentation is a further important direction in which HCI research is needed to support the design process. Documentation needs of many kinds and at many levels arise in the production of a new interactive system. Computer scientists need to find effective ways to apply our their technology (e.g., hypertext, information storage and retrieval techniques, multimedia, CD-ROMs, browsing methods) to yield new ways of delivering information about our systems, making this information more accessible and more useful.

An important concept in writing manuals for users is that information on handling error conditions must be a part of the context of the task in which the error can occur (Carroll, 1984). This is in contrast to the typical error-free treatment of task performance instructions with information on handling errors remotely located and difficult to associate with a specific task. There is a need for methods to embed error handling subtasks directly into task descriptions for user documentation. The design process, where considerations of error-related behavior are made, is the proper place to capture this information.

Another aspect of the documentation issue is production of the documentation itself. The iterative interface development process leads to design changes, often frequent and rapid changes. The effort to maintain currency of good user documentation can be overwhelming in the context of this kind of fast changing development process. There is a need for a fast-track connection, possibly through semi-automated methods for producing user documentation directly from designs. To accomplish this, the technique used to represent the interaction component of the interface must include task descriptions and screen pictures/scenarios. Further, task descriptions that document the interface design must include more information useful for end-users (e.g., strategy information about why a user does particular actions). Such a semi-automated process can lead to at least an interim form of user documentation that is a complete and correct skeleton of technical content as it is known at a given point in the design. Recognizing that automatically produced user documentation will not meet final requirements for writing style and usability, at points when the iterative change cycle is more stable and higher quality documentation is desired, technical writers can craft the skeletal documentation into a more finished product. Having both system and documentation translated from the same design representation can lend high integrity to the match between user documentation and the system. The connection between this kind of documentation and design representation techniques is clear, and those techniques are the subject of the next section.

5. DESIGN SPECIFICATION AND REPRESENTATION

5.1 Introduction

Like design itself, representation or specification (the process of recording the design for analysis and communication among developer roles) is needed both for the interaction component of the interface and for the interface software: a *behavioral* representation for the interface and a *constructional* representation for the software used to build the interface (Hartson, Siochi, & Hix, 1990c). Design representation is important to the interface development process because it is the means for understanding and communicating the design among team members/roles, and it is where much of the analysis of the design is done. For example, some kinds of analysis can be accomplished by automatic processing of a design representation expressed in a formal notation (Reisner, 1981). As another example, mentioned earlier in the section on the iterative development process, psychological design rationale can be included in design representations for the purpose of analyzing psychological claims embedded in interface designs.

The topic of design representation can be difficult to sort out, because there are so many different kinds of representation techniques, each for a different purpose. Each type of representation uses its own perspective to describe the same thing: what is happening in the user interface. For example, suppose the user clicks the mouse button when the cursor is on an icon. This is seen from a behavioral view as a user action within a task, but in a constructional view this is an input event received by the system, and in an implementation view this can be seen as something that fulfills a condition that triggers a function within a toolkit widget. When the icon is highlighted, it is seen as perceptual feedback in the behavioral view, and system response output in the constructional view, due to a default function or perhaps a callback in the implementation view.

A storyboard scenario is usually thought of as behavioral, because it depicts a procedure performed by the user. On the other hand, a state transition diagram is constructional because it is based on a view of interaction that casts the system in a role of waiting in some state for an input which, when received from the user, causes a state change. In the past the most common interface representation techniques have been constructional (Green, 1985; Green, 1986; Hill, 1987; Jacob, 1985; Jacob, 1986; Olsen & Dempsey, 1983; Sibert, Hurley, & Bleser, 1988; Wasserman & Shewmake, 1985; Yunten & Hartson, 1985).

Task analysis methods are behavioral and, therefore, have the potential to be used for design representation. However, they were not originally intended for this purpose, and would require some adaptation to be suitable. For example, hierarchical task decomposition used in task analysis does not usually carry procedural or temporal information. Operation sequence diagrams represent only sample instances of interaction. Scenarios are usually employed only informally to get an early impression of look, feel, and behavior, but it is possible that they could be modified to play a more formal part in design representation (Hartson, Hix, & Kraly, 1990b). Other behavioral representation schemes include GOMS (Card, et al., 1983), the Command Language Grammar (Moran, 1981), TAG (Green, 1989), the keystroke model (Card, et al., 1980), action grammars (Reisner, 1981), and the work of Kieras and Polson (1985). These, however, are intended more for analysis (e.g., predicting user performance of existing designs), than for capturing designs as they are developed. Nonetheless, some of these techniques have seen some use for behavioral design representation (e.g., GOMS).

At Virginia Tech we have been developing and using a behavioral representation technique called the User Action Notation (UAN) for design representation (Hartson, et al., 1990c). The UAN is a task-oriented notation that describes the behavior of the user and the interface during their cooperative performance of a task. An interface is represented as a quasi-hierarchical structure of asynchronous tasks. User actions and tasks are combined with temporal relations such as sequencing, waiting, interrupting, interleaving, and concurrency to describe allowable user behavior (Hartson & Gray, 1990a).

Design rationale has also been recognized as an important part of the design to capture as part of its documentation. If all team members cannot be in the same room at the same time, there is a need to communicate more than the outcomes of the design decisions; there is an additional need to communicate the reasons why decisions were made—a need than goes beyond the design team to those responsible for maintenance, to avoid repeating wrong design decisions. Documentation of design rationale is also essential for analysis of psychological implications of an interface design.

5.2 Key Research Issues

One of the specific research challenges in this area is to find ways to record design ideas with a representation technique that is interleaved with and closely tied to the creative mental design process. How can we make it easy for designers to read and create design representations without distracting mental attention from the creative aspects of design itself? Representation must be made to serve design, not interfere with it. Among the research issues to be addressed is the search for entirely new approaches to design representation. These techniques must be based on sound principles (e.g., abstraction, step-wise refinement) that have been successfully applied in other development domains.

Non-trivial designs are necessarily complex, but some complexity can be controlled with well applied methods of abstraction and some can be overcome with training (interface development certainly is a specialized skill that will require training). It has been shown that some kinds of design representations are easier to understand than others (e.g., graphical over text where possible). The overall development process would benefit from appropriate representation techniques.

Supporting software tools are crucial here. We have tools for prototyping and for constructing, but not really tools for representing designs in the behavioral domain (i.e., interface designs independent of software, software toolkits, and programming aspects). The leverage needed to make this design representation process less laborious, time consuming, and distracting from design can be had only through software support tools.

In addition to notations and techniques, there is a need for good *methods*. In our research group we know how to apply our representation techniques because we invented them. But we are often struck by how much trouble other people have in applying these techniques in the right way, especially in the context of the overall design of a large system.

In this paper, we have given much discussion to behavioral techniques. That is, in part, because they have received so little attention in HCI research. Yet it is in the behavioral domain that the interface designer and evaluator do their work. In addition, of course, the HCI community needs to give attention to highly efficient and powerfully expressive constructional representation techniques, especially those suitable for asynchronous interaction styles.

In our own work on representation techniques, we are very clear on at least one thing: no single technique, view, or perspective is sufficient for all purposes. Specific research is needed to answer questions about how representation of a design can be distributed over various different techniques to support the many developers and related roles that require access to the same design but with very different views or perspectives (e.g., appearance, behavior, temporal aspects, feedback perspective, etc.).

The architectural design of a building provides a useful analogy for this need for multiple views. First, there is an obvious need for a side view, front view, back view, plan view, perspective, each of which is a different view of the same thing, viewed by the builders for a possibly different purpose. But more importantly, the plumber needs a view that shows the plumbing, the electrician needs a view that shows the wiring, and someone else needs a view that shows the heating, air conditioning, and ventilation. These are all *projections* of one single design, each oriented toward a specific different need in the process of developing the building. This is related to the concept of having different views of the same object (or different parts of the same object) displayed in different windows in a user interface. As changes occur to the object, the changes can be seen occurring simultaneously in different ways in each of the views. The broad range of views necessary for a complete and highly usable interface design representation will surely demand powerful tool support within the interface development environment.

There are also key research issues in psychological design rationale as a part of design representation. These involve articulating psychological theories—particularly claims about the user and the tasks—that appear in the design. Representation is important here because it embodies the design and is the medium through which one can ask questions about the design. For this purpose design representation can be driven by user interaction scenarios. The designer constructs a set of user interaction scenarios, perhaps guided by predecessor artifacts, needs analysis, etc. Then the scenarios are developed in detail, in terms of the kinds of events and interactions that will occur, things that the user must do to take the task to its conclusion. This is a concrete representation that can serve as the basis for such analysis.

6. USABILITY SPECIFICATION

6.1 Introduction

An iterative development cycle, having no apparent end, can rightfully cause management concerns about control. It is potentially a formula for chaos to have no fixed order of activities and no milestone marks at the end of phases. How does one know when the process is completed? Answers are emerging from a method called usability engineering (Whiteside, Bennett, & Holtzblatt, 1988). Usability specifications (Carroll, et al., 1985), specific measurable criteria for user performance and satisfaction, are stated during early development activities. Management can approve parts of the design as user evaluation shows corresponding usability specifications are met. Convergence to improved design is served by techniques such as impact analysis (Good, Spine, Whiteside, & George, 1986), which involves measuring time spent by user subjects with interface problems, focusing attention on parts of the interface design that detracted from meeting usability specifications. We have successfully applied many of these techniques in our own development projects and have found them effective.

6.2 *Key Research Issues*

Key research issues in improving the process of creating and applying usability specifications in the interface development process begin with acquiring a better understanding of the true nature of usability itself. This could lead to better ways of formulating usability specifications to address more directly the usability problem in the practical arena of product development.

User performance is a big economic factor, reflected in the fact that most real world usability specifications are centered on objective aspects of user task performance such as timing and error rates. But in the consumer market place it is hard to escape the conclusion, based on how vendors spend their resources, that subjective user preferences are also extremely important. Yet we do not seem to know how to specify and obtain reliable and useful measures of user preferences early enough to drive the design process, short of spending enormous amounts of money on market surveys. It does not appear that any other consumer industry (such as the automobile industry) really knows this, either, but in those industries the problem is simpler because the products (e.g., automobiles or cameras) are much easier to identify and are in more constrained domains.

7. PROTOTYPING

7.1 *Introduction*

Early literature contains descriptions of several approaches to user interface prototyping, including IDS (Hanau & Lenorovitz, 1980), ACT/1 (Mason & Carey, 1981), FLAIR (Wong & Reid, 1982), and RAPID/USE (Wasserman, et al., 1985). A survey of rapid prototyping is given in (Hartson & Smith, 1991). User interface development is based on user evaluation, and the information obtained in evaluation is required before much time and effort are invested in design and hopefully almost none invested in implementation. Thus, in order to have something to evaluate at an early stage, a prototype is used in place of the real system.

Therefore, the closer a prototype can come to the appearance and behavior of the real system, the more effective its contribution to the development process. Many existing prototyping tools, however, are built around a specific look and feel, making it difficult to deviate from the established interaction styles. Current approaches to prototyping are also limited in the range of applications that can be simulated.

7.2 Key Research Issues

Prototyping work has been viewed by many as pedestrian, but as almost any developer who has used a prototype in the development process will attest, there is a significant need for advancement of the state of the art. An important direction for progress is one that eliminates the distinction between tools for rapid prototyping and tools for development of the real system. The goal is graceful evolution from early prototypes to a version of the real system without the discontinuity of a throw-away prototype. Also, if the same tool is used for both prototyping and development, a better match for look and feel and functionality results between the prototype and the real system. The prototype *is* the real system, just in an early stage of development. In software support tools the ability to both interpret and compile the interface description is necessary. These tools also need aids for mocking up the appearance of complex functionality early on. For example, ways are needed to provide sample records to be retrieved by an information system in a user task that involves retrieval. The ability to connect a prototype to existing application functionality is also important, to provide not just interface façades, but whole-system prototypes that can be tested with real user tasks.

8. FORMATIVE EVALUATION

8.1 Introduction

Formative evaluation means testing *as part of the development process.* As such, formative evaluation is a pivotal part of the development lifecycle, closing the loop for iteration. Evaluation can be empirical (by user testing) or analytic. Empirical results can be qualitative (e.g., critical incident analysis or protocol taking) or quantitative (e.g., numeric data), objective (e.g., task performance metrics) or subjective (e.g., user satisfaction measures). And testing must escape the confines of the laboratory. Thomas and Kellogg (1989) warn that laboratory testing, while a productive step toward addressing usability concerns, is not enough. In particular, laboratory testing does not lead to the kind of "ecological validity" that comes from rich, qualitative observations in real work contexts with real users doing real tasks.

Analytic evaluation can depend directly on the design representation, processing it according to a model (e.g., of consistency) and is used to predict user performance without testing. Of course, testing is required to validate the model. Reisner's (1981) paper on the Robart system is definitive of this approach.

A cognitive walk-through for theory-based design (Lewis, Polson, Wharton, & Rieman, 1990) is an evaluation method that focuses attention on a moment-by-moment analysis of the user's mental processes. Attempts have been made to support that analysis with some theoretical conception of how information presented by the system and background knowledge interact. The "artifact as theory" concept discussed under iterative refinement in Section 2 is, of course, also a relevant part of the current state of the art here in formative evaluation.

8.2 Key Research Issues

Although there is already a lot of discussion and motivation for this in the literature, this is one of the areas that is ripe for new research in both theory and practice. We do not yet know enough about what makes good formative evaluation work and bad formative evaluation not work. At the moment, formative evaluation may be more an art than a science. There is a pressing need for new, more precise, techniques that are effective in quickly assessing user performance. Many existing evaluation techniques were contrived to fit a certain development situation in a certain application domain. We need ways to formalize and codify evaluation processes so they can be brought into an evaluation methodology applicable across a broader spectrum of situations. There is especially a need for techniques that can assign credit and blame, pinpointing *why* user performance is not up to expected levels in terms of specific interface design flaws and shortcomings (and suggested improvements for same). Methods are also needed to manage the problem of prioritizing, for optimum allocation of finite future redesign effort, design problems discovered in evaluation.

One key future research direction involves analytic evaluation, which would benefit greatly from solutions to some difficult problems. Once a task description is obtained from task analysis, there are analysis techniques that have the potential for predictive capability. It is unlikely that we will ever be able to even come close to automating all analysis of interface designs. However, Reisner's early work (Reisner, 1981) has shown that there are classes of design flaws amenable to analysis based on computer processing of design representations. Because user actions at the articulatory level are repeated in different combinations to make up all higher levels of abstraction, actions at this level are especially suitable for analysis techniques that predict and improve user performance. It is a significant technical challenge to extend this type of analysis work, but the benefits could be valuable, especially for real world designs that are too large for an individual to understand at one time. For example, there are big opportunities for other kinds of analysis to use the task descriptions to reveal (and thus support in the design) user cognitive and information needs at each step in the performance of a task, but these analyses would require capturing much more information about what users think about the tasks, how they make decisions, how they make use of information, etc. Another kind of analysis using task descriptions could be based on automatic processing of design representations to identify problems with consistency, ambiguity, and implementability

Research in the area of formative evaluation needs to focus on iterative evaluation techniques that, in fact, lead to convergence on a good, or at least improved, design (as compared to a benchmark). *Ad hoc* random probing is not sufficient in a product development environment. Nonetheless, many approaches currently used in the real world are unstructured and the resulting interface designs sometimes do not converge on high usability. There are even cases where usability has decreased with each iteration. Thus, a key research issue in this area is to find a way to do evaluation so that it yields more than just a figure of merit that says how well the product works, but one that also reveals why it does not work as well as desired. This can be done in many ways. The evaluator must look at strategies and task context of the user. How did people who did well approach the task? What kind of things were they doing when they succeeded? Or when they were slow or inaccurate? Of which features did heavy use correlate with good or bad performance? It often looks obvious afterwards, but no structured method really exists to ferret out such information during evaluation.

As an example of why just passive observation is inadequate, consider the case where a user makes a wrong choice from a menu. We can observe that, but we cannot observe the reason why. Yet it is the reason *why* that must be addressed in order to make the right adjustment to the design. On one hand, the user may have the right idea of what is supposed to be done, but the menu is misleading. On the other hand the menu may be clear, but the user has the wrong idea of what to do. Sometimes verbal protocol taken during formative evaluation can sort some of this out, but it is not always the case that the user will know what the real problem is, or will say the right thing for the evaluater to identify the cause of the problem.

Perhaps a structured combination of techniques is required. Maybe a critical incident-driven principle-based structured verbal protocol method can be used to address the question of *why*. Critical incident techniques will identify situations where the question arises (e.g., the user did not make the right menu choice). This method would then be designed to follow up immediately with the appropriate principle-based questions to ferret out the reason behind the user error and help interpret the reason specifically in terms of redesign needs. The technique involves conducting the verbal protocol in a specific structured way, as opposed to random eavesdropping on the user, with the specific goal of finding the reason behind the user problem. This permits a line of traceability from user performance back through representation, design, and even to task analysis. The cognitive walkthrough method is related to this line of principle-based reasoning about the design (Lewis, et al., 1990).

Recent work by NYNEX and CMU (Gray, John, Stuart, Lawrence, & Atwood, 1990) exemplifies one direction that shows promise in attacking this research goal of providing specific feedback from evaluation to design. This work combines GOMS techniques with critical path methodology to allow sensitivity analysis of designs by predicting the difference in user performance for two different designs, and compares it to performance measured in a field trial. The cognitive, perceptual, verbal, and motor actions for a task, plus response time of the system, are represented on a critical path timing chart. The method allows a determination of the effect on user performance, given a change in task procedures or in keyboard layout, that voice recognition is added somewhere, or that the system response time is changed. The effect may be greater or smaller, depending on whether the part of the task affected by the change is on a critical path. This provides an explicit connection between design and user performance. The method, now being validated, has potential for application to other tasks and is ripe for use in real world applications.

9. MODELING

9.1 Introduction

Modeling and theory go hand-in-hand, providing the foundation upon which we soundly build our techniques and tools. As scientific tools in HCI, the topics of modeling and theory are treated in a separate paper. The scope of interest in modeling here is limited to its use specifically in the development process. There are two kinds of models that are supportive in this capacity: models of the system and/or software architecture and models of interaction.

Architectural models are used to structure systems into more manageable components for modularity and complexity control (e.g., (Green, 1985; Pfaff, 1985)), often resulting in some kind of separation between the interface and computational components. The gains in modularity are somewhat offset by a loss of cohesion, leading to further modeling requirements to treat logical control and communication across module boundaries (Coutaz, 1985; Coutaz & Balbo, 1991; Hartson, 1989; Hurley & Sibert, 1989).

While modeling of human-computer interaction will have some overlap with the paper on "Theory and Taxonomy of HCI Models," it is also important here because it shows up directly in methods (and tools) for interface development. Modeling of interaction is used to identify, define, and characterize each of the basic components (entities, objects, artifacts, phenomena, actions, parameters, etc.) of interaction and to guide the designer in building up the interface components, providing an orderly way of approaching the various features and abstractions within a structured framework (Foley, Gibbs, Kim, & Kovacevic, 1988).

9.2 Key Research Issues

HCI work so far in modeling to support interface development has been limited and not widely transferred to practitioners. Further work in this area is, indeed, a key research direction. The alternative is to continue interface design as an unorganized attack that can easily lead to an overwhelming sea of detail. In contrast, a model-based development process can make use of built-in semantic knowledge of the constituent parts and pieces of an interface and can direct the designer's attention to these in an orderly manner.

Other kinds of semantic models of interaction are also needed to capture the meaning of user actions in terms of their effects on interface and application objects and deal with objects, relationships, constructional connections to the application, and behavioral connections to the task. These models are needed to bridge the gap between the task-oriented behavioral world of the user and the constructional object- and toolkit-oriented world of interface software, and beyond into the non-interface world of a system's computational software. The meaning of a user action can, for example, be expressed in terms of the comprehensive effects of that user action. For an action on a complex interface object, the semantics can involve complicated interaction among its components. For example, consider the dialogue box for specifying a file to open in a Macintosh application. Operating within this dialogue box, the user can take actions that have effects on files, directories, scrolled lists, selection, ejection of disks, switching of active volumes, etc. Linguistic models (used mostly with the older typed command language interaction styles) identify and process tokens and other information occurring at semantic, syntactic, lexical, and pragmatic levels (Foley & Wallace, 1974). Now there is a continuing research need for better models of asynchronous interaction styles.

10. CONNECTIONS TO OTHER PAPERS

Just as we do with interface designs, we have divided the large topic area of future HCI research into subareas to achieve some degree of modularity. And as it happens with interface designs, this also leads to problems of where to draw lines that separate the modules and how to express connections among them. This section provides some cross references to the subareas of other papers in hopes of showing relationships among the subjects involved.

10.1 Software Engineering

At the beginning of this paper, we claim that the goal of HCI work is to help produce better quality interactive systems. We know the user interface has much to do with the quality of a system, in particular its usability. But there is obviously more to a system than its interface, and the discipline of software engineering shares the goal of better systems with HCI, which it pursues via many directions, including requirements specification, formal program verification, reusability, and maintainability.

There are HCI concerns within software development as well as for software end-users, including usability issues for software tools (e.g., CASE tools) in development environments. Also developers of the non-interface (computational) component of application software must grapple with the connection to the user interface, seeing the problem from their side of the system.

Since the design of the interaction component of a user interface maps into a specification for design of interface software, there are also strong software roles in interface development. These include the need for software design and implementation models to structure this mapping, design and implementation of interface software, use of toolkits, and development of new and more suitable toolkits. In particular there is a need to provide tools and toolkits (e.g., I/O support libraries) that offer a better match to the interaction styles and devices used and needed by interface designers. And, again, the maintenance issue arises; how do changes to the interface affect the software?

10.2 Software Interface Development Tools

Interface development tools bear a very strong relationship to the development activities discussed here. We need tools to support the full range of interface development activities, not just construction of interface software. If tools provide the support environment for the process, tool developers must understand the process. Tool developers have not always been tuned to the whole process (e.g., software engineers often overlook task analysis).

Development activities occurring later in the process tend to be better understood and are easier to support with software tools. This trend is easy to see in today's CASE (Computer Aided Software Engineering) tools. Tools are also needed to support early activities of interface development are needed. There are few, if any, interface development tools to support user needs analysis, requirements analysis, and task analysis. Perhaps some of these can be adapted from existing software engineering tools oriented to up-front systems analysis. There is a similar lack of tools to support the creative parts of design and tools to capture design representations. We begin to find available tools only when we consider the need to move the representation to a prototype and then to move to the real system. Here is, for example, where most of the work on UIMS has been directed so far. However, many of the tools we do have for prototyping and implementation are still rudimentary and often difficult to use. Also, since many of these tools are now being used to generate interface code, some of the design of the implementation model for interfaces is now being rolled back into the design of those tools. Even prototyping tools, which are probably the most commonly available of all interface development tools, need to be much more powerful and need to be model-based.

Finally, as we pointed out in the section on design representation, representation must be made to serve design and not interfere with it. The kind of complexity control needed to accomplish this can be provided only by interactive tools. In particular, there is a need for tools to represent designs in the behavioral domain (i.e., interaction designs independent of software, software toolkits, and programming aspects). The leverage needed to make this design representation process natural and not too laborious will rely heavily on interface development tools.

10.3 Modeling and Theory

There are many kinds of modeling and theory work. The paper addressing the topic of "Theory and Taxonomy of HCI Models" is concerned with models on the science side of HCI, while in this paper we are concerned with models on the development side. The difference is that the approach in this paper has to do with modeling as applied within a project for the development of a specific application product. The approach of the other paper is more theoretical, looking at models for furthering our understanding of HCI in general.

One way research in the area of theoretical models can help with development is by providing models of application domains (e.g., word processing, scientific applications) to fold into the design process. And, of course, we always look to the theoretical work in search of enabling concepts and theories, to find missing links, to find what we need to make it all work.

10.4 Computer Supported Cooperative Work (CSCW)

While there will always be individual development efforts, there is a trend within larger development efforts toward a team approach where the team is composed of various, sometimes specialized, roles. This kind of closely-couple collaborative work is the subject of CSCW research and much of that work centers on communication of various kinds. Documentation (in the broadest sense and including representation) of the design is central to this communication. If only two or three developers are involved, documentation can be somewhat informal and can be supplemented by informal communication channels. For a larger team effort communication needs dictate that documentation must be more formal and more complete. Also, since the documentation is an account of the state of the design at any given moment, practical methods are used for version control to determine who can "check out" what parts of the design documents and what kind of changes are authorized by whom. Checking out a part of the design document temporarily locks out any changes from other team members. Further, since the process is iterative, a group member might get the same document back several times, with mandates for new changes. This leads to the need for formal ways for the group as a whole to agree on when they are done with some piece of the design.

Another concept, called simultaneous engineering (a concept taken from the automotive industry), requires all interested parties to work together in the same room (either physically or remotely via teleconferencing) to speed up the communication process. Situations where questions must be answered before design decisions can be made or where rapid feedback of constraints and problems (e.g., cases where user needs or implementation costs rule out a design feature) is needed can bring about more rapid convergence. It prevents someone from going too far in a non-productive direction before reaction from the others will get the design back on track. Specific work has begun, in fact, by CSCW researchers on software engineering and system development as an application area of CSCW. We look to further work in this area for results to apply in the team development environment for user interfaces.

11. INFRASTRUCTURE NEEDS

Note: Since there is a separate paper on infrastructure needs, this can be considered as an input to that.

User interface development, like interactive system development of which it is part, is an in-the-large process. Research in this area will necessarily involve empirical observation and measurement of in-the-large activities. Very few such large scale project-oriented empirical studies have been done (Alavi, 1984; Boehm, Gray, & Seewaldt, 1984), because of the resources required and because of the scarcity of opportunities within real projects. It is not just a question of money and other resources; in the real world where timing is critical in project management, researchers will not find enthusiasm for any study that could interfere with an already difficult, crammed, and sometimes stressful development process and put it in danger of being even slightly delayed. There is great economic pressure not to meddle with an established real world process, however inefficient and ineffective it may be.

The success of a large project-based study, then, depends on making the right connections with people who are responsible for project management, cultivating an understanding of the need for and potential payoff from such a study, and convincing people that the study will not interfere with getting the product to market on time. For university researchers, at least, these requirements mean a kind of university/industry cooperation rarely possible these days.

Other infrastructure requirements related to future HCI research include support of HCI research by computer science departments and universities themselves. Even a modest usability testing laboratory needs hardware, software, network connections, soundproofing, video cameras, video monitors and mixers, money to pay subjects, and—the biggest need of all—staff for building and maintaining test bed systems and prototypes to test.

12. THE FUTURE

We perceive the present state of the art of user interface development as being one in which there are many theories, methods, and techniques emerging and beginning to be tested for attacking isolated development steps. These isolated steps include task analysis, design, prototyping, evaluation, and iterating to improve the design, for example. However, the overall process is still often quite ad hoc and not well integrated.

In the future we need a more formal, structured basis for progress in HCI research. This kind of growth and maturation is natural to the lifecycle of a new technology; the same thing happened as structured programming evolved in the early days of software engineering. We will see steady progress as we understand more about the problem. Hopefully, we will also see quantum leaps that will carry us forward even more decisively. We are glad to be part of it.

ACKNOWLEDGEMENTS

The author wishes to thank the following people for inputs to the paper and for reading and commenting on various drafts. Each of these people contributed significantly to the ideas and words herein.

Within the workshop group:
Deborah Boehm-Davis

Clayton Lewis
John Carroll
Thomas Landauer
Judith Reitman-Olson
John Hestenes
Jon McKeeby (student aide)

Outside the workshop group:
Deborah Hix
Roger Ehrich

REFERENCES

Alavi, M. (1984). An Assessment of the Prototyping Approach to Information Systems Development. *Communications of the ACM, 27*(6), 556-563.

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer, 21*(3), 61-72.

Boehm, B. W., Gray, T. E., & Seewaldt, T. (1984). Prototyping vs. Specification: A Multi-Project Experiment. *Proceedings of Seventh international Conference on Software Engineering*, New York: ACM & IEEE, 473-484.

Card, S. K., & Moran, T. P. (1980). The Keystroke-Level Model for User Performance Time with Interactive Systems. *Communications of the ACM, 23*, 396-410.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The Psychology of Human-Computer Interaction* . Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Carey, T. (1982). User Differences in Interface Design. *IEEE Computer, 15*(11), 14-20.

Carroll, J. M. (1984). Minimalist Design for Active Users. *Proceedings of Human-Computer Interaction—Interact '84*, Amsterdam: North-Holland, 39-44.

Carroll, J. M. (1990). Infinite Detail and Emulation in an Ontologically Minimized HCI. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 321-327.

Carroll, J. M., & Kellogg, W. A. (1989). Artifacts as Theory-Nexus: Hermeneutics Meets Theory-Based Design. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 7-14.

Carroll, J. M., & Rosson, M. b. (1985). Usability Specifications as a Tool in Iterative Development. In H. R. Hartson (Ed.), *Advances in Human-Computer Interaction* (pp. 1-28). Norwood, NJ: Ablex.

Coutaz, J. (1985). Abstractions for User Interface Design. *IEEE Computer, 18*, 21-34.

Coutaz, J., & Balbo, S. (1991). Applications: A Dimension Space for User Interface Management Systems. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 27-32.

Draper, S. (1989). *Personal communication.*

Draper, S. W., & Norman, D. A. (1985). Software Engineering for User interfaces. *IEEE Transactions on Software Engineering, SE-11*, 252-258.

Fischer, G., Lemke, A. C., Mastaglio, T., & Morch, A. I. (1990). Using Critics to Empower Users. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 337-347.

Foley, J., Gibbs, C., Kim, W., & Kovacevic, S. (1988). A Knowledge-Based User Interface Management System. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 67-72.

Foley, J. D., & Wallace, V. L. (1974). The Art of Natural Graphic Man-Machine Conversation. *Proceedings of the IEEE, 63*(4), 462-471.

Good, M., Spine, T., Whiteside, J., & George, P. (1986). User Derived Impact Analysis as a Tool for Usability Engineering. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 241-246.

Gould, J. D., Boies, S. J., & Lewis, C. (1991). Making Usable, Useful, Productivity-Enhancing Computer Applications. *Communications of the ACM*, *34*(1), 74-85.

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

Gray, W. D., John, B. E., Stuart, R., Lawrence, D., & Atwood, M. (1990). GOMS Meets the Phone Company: Analytic Modeling Applied to Real-World Problems. *Proceedings of INTERACT '90—Third IFIP Conference on Human-Computer Interaction*,

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

Green, M. (1986). A Survey of Three Dialog Models. *ACM Transactions on Graphics*, *5*(3), 244-275.

Green, T. R. G. (1989). Task Action Grammar, presented at the British Computer Society HCI Specialists Group Day Meeting on Task Analysis, May, London. No proceedings.

Hanau, P. R., & Lenorovitz, D. R. (1980). A Prototyping and Simulation Approach to Interactive Computer System Design. *Proceedings of Design Automation Conference*, New York: ACM, 572-578.

Hartson, H. R. (1989). User-Interface Management Control and Communication. *IEEE Software*, *6*(1), 62-70.

Hartson, H. R., & Gray, P. (1990a). Temporal Aspects of Tasks in the User Action Notation, To appear in *Human Computer Interaction.*

Hartson, H. R., & Hix, D. (1989). Toward Empirically Derived Methodologies and Tools for Human-Computer Interface Development. *International Journal of Man-Machine Studies*, *31*, 477-494.

Hartson, H. R., Hix, D., & Kraly, T. M. (1990b). Developing Human-Computer Interface Models and Representation Techniques. *Software—Practice and Experience*, *20*(5), 425-457.

Hartson, H. R., Siochi, A. C., & Hix, D. (1990c). The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs. *ACM Transactions on Information Systems*, , Hartson, H. R., & Smith, E. C. (1991). Rapid Prototyping in Human-Computer Interface Development. *Interacting with Computers, 3*(1), 51-91.

Hill, R. (1987). Event-Response Systems — A Technique for Specifying Multi-Threaded Dialogues. *Proceedings of CHI+GI Conference on Human Factors in Computing Systems*, New York: ACM, 241-248.

Hurley, W. D., & Sibert, J. L. (1989). Modeling User Interface-Application Interactions. *IEEE Software, 6*(1), 71-77.

Jacob, R. J. K. (1985). An Executable Specification Technique for Describing Human-Computer Interaction. In H. R. Hartson (Ed.), *Advances in Human-Computer Interaction* (pp. 211-242). Norwood, NJ: Ablex.

Jacob, R. J. K. (1986). A Specification Language for Direct Manipulation User Interfaces. *ACM Transactions on Graphics, 5*(4), 283-317.

Kieras, D., & Polson, P. G. (1985). An Approach to the Formal Analysis of User Complexity. *International Journal of Man-Machine Studies, 22*, 365-394.

Lewis, C., Polson, P., Wharton, C., & Rieman, J. (1990). Testing a Walkthrough Methodology for Theory-Based Design of Walk-up-and-Use Interfaces. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 235-241.

Mackinlay, J., Card, S. K., & Robertson, G. G. (1990). A Semantic Analysis of the Design Space of Input Devices. *5*,

Mason, R. E. A., & Carey, T. T. (1981). Productivity Experiences with a Scenario Tool. *Proceedings of COMPCON '81*, New York: IEEE, 106-111.

Moran, T. P. (1981). The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems. *International Journal of Man-Machine Studies, 15*, 3-51.

Olsen, D. R., Jr., & Dempsey, E. P. (1983). Syngraph: A Graphical User Interface Generator. *Computer Graphics, 17*(3), 43-50.

Pfaff, G. (1985). *User Interface Management Systems*. Berlin: Springer-Verlag, Reisner, P. (1981). Formal Grammar and Human Factors Design of an Interactive Graphics System. *IEEE Transactions on Software Engineering, SE-7*, 229-240.

Rosson, M. B., Maass, S., & Kellogg, W. A. (1987). Designing for Designers: An Analysis of Design Practice in the Real World. *Proceedings of CHI+GI Conference on Human Factor in Computing Systems*, New York: ACM, 137-142.

Rosson, M. B., Maass, S., & Kellogg, W. A. (1988). The Designer as User: Building Requirements for Design Tools from Design Practice. *Communications of the ACM, 31*(31), 1288-1298.

Sibert, J. L., Hurley, W. D., & Bleser, T. W. (1988). Design and Implementation of an Object-Oriented User Interface Management System. In H. R. Hartson (Ed.), *Advances in Human-Computer Interaction* (pp. 175-213). Norwood, NJ: Ablex.

Smith, S. L., & Mosier, J. N. (1986). *Guidelines for Designing User Interface Software* (ESD-TR-86-278/MTR 10090). The MITRE Corporation, Bedford, Mass.

Thomas, J. C., & Kellogg, W. A. (1989). Minimizing Ecological Gaps in Interface Design. *IEEE Software, 6*(1), 78-86.

Wasserman, A. I., & Shewmake, D. T. (1985). The Role of Prototypes in the User Software Engineering Methodology. In H. R. Hartson (Ed.), *Advances in Human-Computer Interaction* (pp. 191-210). Norwood, NJ: Ablex.

Whiteside, J., Bennett, J., & Holtzblatt, K. (1988). Usability Engineering: Our Experience and Evolution. In M. Helander (Ed.), *Handbook of Humna-Computer Interaction* (pp. 791-817). Amsterdam: Elsevier North-Holland.

Wixon, D., Holtzblatt, K., & Knox, S. (1990). Contextual Design: An Emergent View of System Design. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 329-336.

Wong, P. C. S., & Reid, E. R. (1982). FLAIR—User Interface Dialogue Design Tool. *SIGGRAPH Computer Graphics, 16*(3), 87-98.

Yunten, T., & Hartson, H. R. (1985). A SUPERvisory Methodology And Notation (SUPERMAN) for Human-Computer System Development. In H. R. Hartson (Ed.), *Advances in Human-Computer Interaction* (pp. 243-281). Norwood, NJ: Ablex.