Formative Evaluation: Ensuring Usability in User Interfaces

Deborah Hix and H. Rex Hartson

TR 92-60

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

December 23, 1992

FORMATIVE EVALUATION: ENSURING USABILITY IN USER INTERFACES

DEBORAH HIX & H. REX HARTSON

Department of Computer Science
Virginia Tech
Blacksburg VA 24061 USA

internet: *lastname*@vtopus.cs.vt.edu

703/231-6199 *or* 703/231-4857

ABSTRACT

Ensuring usability has become a key goal of interactive system development, as developers have begun to realize that it matters little how effectively an interactive system can compute if human users cannot communicate effectively with the system. In response to this need for increased usability, interactive system developers have realized the necessity for techniques to evaluate user interface design—to determine existing levels of usability and to identify problems to be solved in order to improve usability. In this paper we will discuss what we have found to be two main types of *formative user interface evaluation*: analytic and empirical. Both these types occur as part of the development process. We do not attempt to survey all approaches to either of these types of formative evaluation, but rather to offer a sampling of some approaches that have been found (by us and by others) to be useful in ensuring usability. We give only an overview of analytic methods, and then focus on empirical methods. We conclude with some of our observations on future trends in user interface evaluation.

FORMATIVE EVALUATION: ENSURING USABILITY IN USER INTERFACES

DEBORAH HIX & H. REX HARTSON

1. INTRODUCTION

Stories abound about how *not* to evaluate a user interface. One of our favorites came from an undergraduate student telling us about her summer job. She was hired by a major government contractor to do some implementation on a portion of a huge interactive system. On her first day, the module of the project she was to work on was not quite ready for her to begin. In an attempt to find something for her to do for the week or so until the module was ready, her supervisor decided, on the spur of the moment, that the student could evaluate the user interface! Her instructions were to "Go play around with this thing and tell us what to fix." This company had planned no evaluation of the user interface for a multi-million dollar interactive system. In fact, the only evaluation the first version had (before it got to people who had to use it) was from this rising junior computer science major who knew absolutely nothing about human-computer interaction design and evaluation. She knew little about what the system was supposed to do, and even less about evaluating an interface. We were relieved (and not a little amused) to hear that the contractor did formally include interface evaluation in its development plan for subsequent versions, because the first release of this system was such a disaster in terms of its usability.

Only a decade ago, evaluation of most interactive systems was much like the scenario just described: ad hoc at best, nonexistent at worst. Users typically saw the system for the first time when it was fully implemented and often all the way to being the "final" deployable product. This method produced the horrifying, unusable user interfaces that proliferated into all application areas. Now, however, developers are finally realizing that the bottom line is: *Users are going to evaluate the interface sooner or later*—either correctly, in-house, using the proper techniques and under the appropriate conditions; or after it is in the field, when it is probably too late to make many effective modifications. For everyone's benefit, the sooner evaluation is done, the better.

Ensuring usability has become a key goal of interactive system development, as developers have also begun to realize that it matters little how well an interactive system can compute if human users cannot communicate effectively with the system. In response to this need for increased usability, interactive system developers have realized the necessity for techniques to evaluate user interface design—to determine existing levels of usability and to identify problems to be solved in order to improve usability. In this paper we will discuss what we have found to be two main types of *formative user interface evaluation*: analytic and empirical. Both these types occur as part of the development process. We do not attempt to survey all approaches to either of these types of formative evaluation, but rather we offer a sampling of some approaches that have been found (by us and by others) to be useful in ensuring usability. In Section 2, we will explain in more detail what our context for this paper is, and what is meant by formative evaluation. In Section 3 we will give only an overview of analytic methods, and then focus, in Section 4, on empirical methods. In Section 5, we summarize, and in Section 6 conclude with some of our observations on future *trends* in user interface evaluation.

2. SETTING THE CONTEXT

2.1. Formative vs. Summative Evaluation

The origins of the terms formative and summative evaluation are traced by Carroll, Singley, and Rosson (1992) to Scriven's (1967) methodological framework of goals and processes for performing evaluation of instructional materials. Dick and Carey (1978) also used these terms in the same context. The terms have since been adopted in the human-computer interaction (HCI) literature (e.g., (Hartson & Hix, 1989; Williges, 1984).

Summative evaluation is evaluation of a design after it is complete, or nearly so. Summative evaluation is often used during field or beta testing, or to compare one product to another. For example, a summative evaluation of two systems, A and B, co) is based on a linear sequence of development phases. However, there are implicit feedback paths from later phases to earlier ones, resulting in cycles that some consider iterative. The spiral life cycle (Boehm, 1988) features explicit cycles, but they are large cycles, passing several times through the same sequence of development phases. On the other hand, user interface development requires more frequent and less global cycling. Often, numerous tight, localized cycles of iteration are required just to achieve the desired level of usability for a single interface feature. In response to the need for this focus on iteration, and correspondingly on evaluation, we have developed a life cycle concept we call the star life cycle (Hartson, et al., 1989), shown in Figure 1.

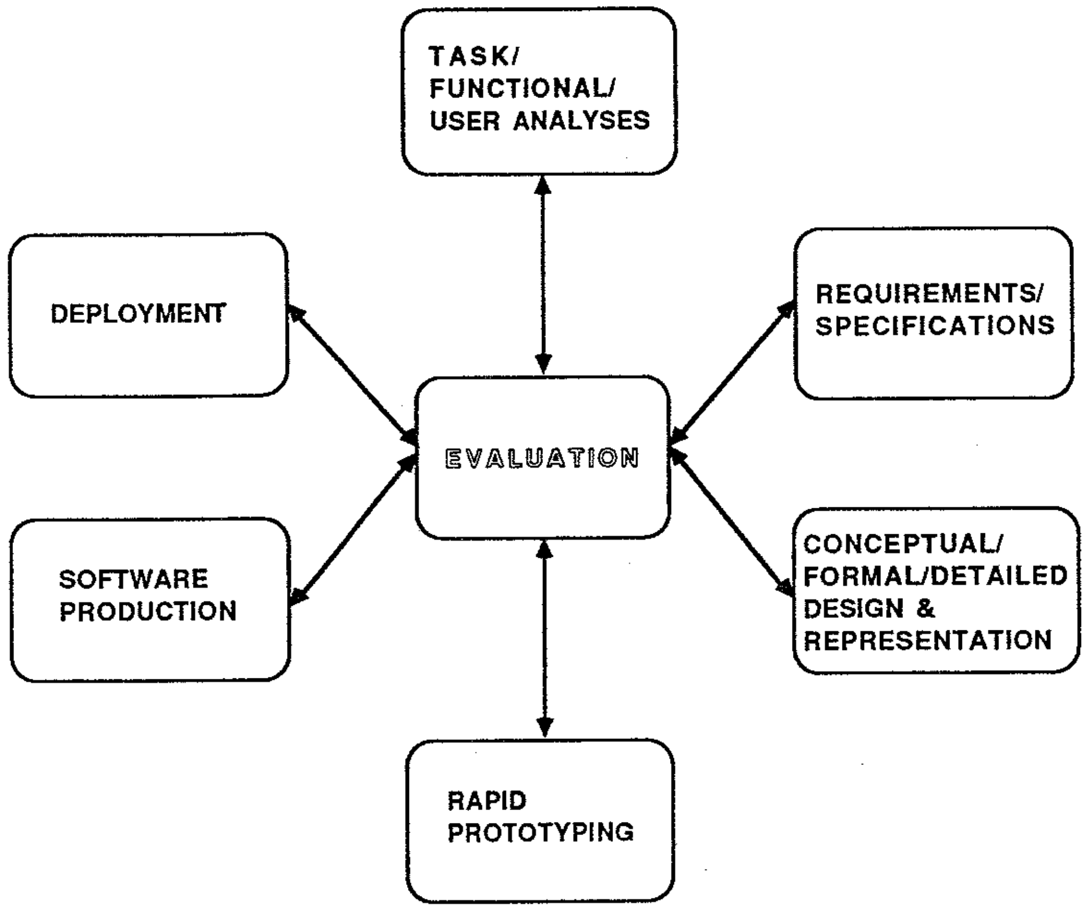


Figure 1. Star life cycle (adapted from (Hartson & Hix, 1989))

The structure—or lack of it, compared to other life cycles—of this star life cycle shows that evaluation (of a kind appropriate to each activity) is to be applied to every user interface development activity (e.g., task/functional/user analyses, requirements/specification), not just to design. It also shows that there is not a single starting point, nor is there a prescribed order for development activities. Simply, each development activity should be followed by evaluation of some sort.

From our own experiences, and in talking with numerous other developers, we have found a rule of thumb to be that an average of three major cycles of formative evaluation, with every cycle followed by iterative redesign, will be completed for each significant version of a design. As we said above, there are also additional very short cycles, to quickly check out a few small issues. The most data generally come from the first major cycle of evaluation. If the process is working properly and the design is improving, later cycles will generate fewer new discoveries and will generally necessitate fewer changes in the design. The first cycle can generate an enormous amount of data, enough to be overwhelming. In Section 4, we will give discuss how to collect and analyze these data in order to meet interface usability goals.

People sometimes mistakenly say that formative evaluation is not as rigorous or as formal as summative evaluation. The distinction between formative and summative evaluation is not in its formality, but rather in the goal of each approach. Summative evaluation does not support the iterative refinement process; waiting until an interface is almost done to evaluate it will not allow much, if any, iterative refinement. It is important that members of the development team, and especially managers, understand this difference. Otherwise, because formative evaluation is not controlled testing and usually does not require many participants (subjects), results of formative evaluation may be discounted as being, for example, too informal, not scientifically rigorous, or not statistically significant. Formative evaluation is, indeed, formal in the sense of having an explicit and well-defined procedure and does result in quantitative data, but is not intended to provide statistical significance. Formative evaluation is a technique used by developers to address the needs of users, and thereby to ensure high usability in an interface.

2.3. Evaluation of What and for What?

An interactive system can be viewed simplistically as shown in Figure 2.

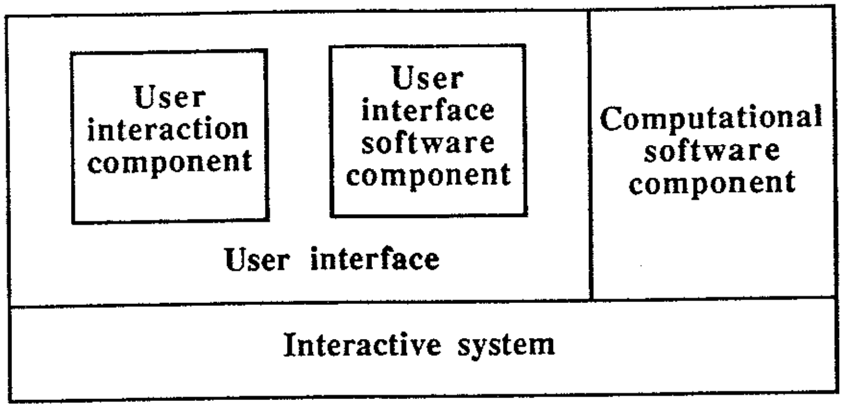


Figure 2. Components of an interactive system

Well-known techniques from software engineering are appropriate for developing the computational (non-interface) software component. Software evaluation can have many goals, including fidelity of implementation to the design, reliability (freedom from bugs), reusability, and so on. Since the user interface software component is, obviously, also software, these same goals apply. Usability is *not* on this list of goals.

So, surprising as it might be to those who have not thought about the distinction, it is not user interface software that is evaluated for usability. Rather, the *interaction design*—what the user sees and hears and does while interacting with the system—is evaluated for usability. The user interaction component includes the icons, text, graphics, audio, video, and devices that are presented to the user and with which the user interacts. User interface software might be the medium through which an interaction design is implemented, executed, observed, and evaluated, but an interaction design might also be manifest by means of a WYSIWYG rapid prototyping tool that does not necessarily produce user interface software in the usual sense. In fact, the user interface could be constructed in firmware or hardware, but it does not matter for our purposes here, since *it is the interaction design that is evaluated for usability*.

Thus, software evaluation techniques are not appropriate for evaluating user interaction designs; to ensure usability, different methods of evaluation are required, and those methods are the topic of this paper.

To further delineate our scope, we mention an attribute of interactive systems that can be measured very early in the development cycle and that is closely related to usability, namely, *user acceptance*. Davis (1989) shows that perceived usefulness and perceived ease of use are fundamental determinants of user acceptance. However, user acceptance metrics, which combine metrics for both usability and functionality of a proposed system, appear to have more direct benefit to management and marketing organizations than to development teams. For example, such metrics do not serve the formative evaluation process because they lack the kind of specific feedback about usability problems necessary for iterative development and design improvement. Therefore, we do not emphasize user acceptance within the scope of this paper.

2.4. Types of Formative Evaluation

As we mentioned earlier, there are (at least) two main types of formative evaluation: analytic and empirical. Many *analytic techniques* use descriptions of the behavioral or user's view of interaction designs based on tasks and/or grammars. These design descriptions are analyzed to detect usability problems early in the development process, perhaps before any implementation, or even prototyping, has been done. Thus, these techniques are often built on predictive models of user behavior and/or performance. These predictive models are sometimes validated by comparing predicted user performance with empirically measured performance (e.g., Reisner's work on Robart (1981), discussed below). Scriven (1967), as described in (Carroll, et al., 1992), uses the term *intrinsic evaluation* for this kind of evaluation technique that studies the inherent structure of a design. These techniques usually involve a very detailed analysis of the design, and often the design rationale, in terms of goals and subgoals.

Empirical techniques for formative evaluation of interaction designs are based on data taken during observed performance testing with users. Typically prototypes are used as the medium for evaluation, in order to identify usability problems early in the development process. Scriven refers to evaluation for rapid detection of design problems as *pay-off evaluation*. This is an appropriately descriptive term for purely empirical approaches to formative evaluation in the area of human-computer interaction, because developers seek an immediate pay-off in terms of design improvements. This is in contrast to some other kinds of empirical work, such as controlled experimentation that may add to our theoretical and principle-based knowledge a little bit at a time.

Both analytic and empirical evaluation in the context of HCI have the same goal: to ensure usability in a user interaction design. However, they are very different techniques for reaching that goal, as we will discuss in the remainder of this paper.

3. ANALYTIC FORMATIVE EVALUATION

3.1. Some Examples of Analytic Evaluation

Before we concentrate on empirical approaches to formative evaluation, we offer a brief overview of some representative approaches to analytic evaluation, including the GOMS model (Card, Moran, & Newell, 1983) the work of Kieras and Polson (1985), the Command Language Grammar (CLG) (Moran, 1981), the keystroke-level model (Card & Moran, 1980), the Reisner Action Language model (1981), and the Task Action Grammar (TAG) (Payne & Green, 1986). These models all are the basis for some sort of analytic evaluation of a user interaction design.

The analytic approach to evaluation of user interaction design is generally based on examining and/or manipulating an abstract representation of that design. The result is a prediction of what would happen if usability were measured empirically with human participants; thus, an analytic approach can be used to compare two alternative designs or two different versions of the same design. As such, analytic techniques are sometime considered a substitute for empirical evaluation, at least at some point in the development cycle. However, *analytic methods must be considered complementary to, and not a complete substitute for, empirical evaluation* (Reisner, 1983). When analytic evaluation is used, it is often used early in the evolution of a design, followed up later with empirical evaluation. By necessity, for the credibility of the prediction, each analytic method must be validated with empirical measurements of human performance.

The GOMS (Goals, Operators, Methods, and Selection rules) model (Card, et al., 1983) supports predictive evaluation through task analysis of an interaction design. GOMS provides a foundation based on psychological theories for purposes of predicting user performance. A goal identifies a task and a method describes how the task is performed, in terms of operators and selection rules. Complex cognitive tasks can be encapsulated within the operators to simplify the modeling. The amount of detail generated in a GOMS interface description allows for a thorough evaluation at a very low level of detail, but the GOMS description can be an enormous, difficult, and often tedious undertaking to produce. As a practical matter, it requires a trained cognitive psychologist exercising expert judgement to produce these details Knowing the difficulty of producing GOMS descriptions and that different GOMS analysts can produce quite different representations, Kieras formulated a practical approach to producing more standardized GOMS descriptions, one that can be used by non-specialists (Kieras, 1988). Despite the complexity of the GOMS approach, there are numerous indications in the literature that it is used, at least to some extent, for developing and evaluating user interfaces.

Using the GOMS model as a basis, Kieras and Polson (1985) have built a formal model to describe user knowledge required for performance of a task and for use of a device. This is a model of cognitive complexity—complexity of interaction as viewed by the user—that results in user performance prediction. This work is sometimes referred to in the literature as cognitive complexity theory. User knowledge is represented in the form of IF-THEN production rules, with the GOMS model providing the content, essentially converting user task performance into a computer program. Complexity measures (e.g., a count of the number of production rules) can be applied to the "programs" to measure the amount and complexity of knowledge required for performing specific tasks. Additionally, generalized state transition networks (GTNs) are used to represent the behavior of devices (Kieras & Polson, 1983). GTN representations for devices, along with production rule representations of users, can be executed together as a single simulation representing both user and devices. The results of simulation are the performance predictions.

The Command Language Grammar (CLG) approach to user modeling (Moran, 1981) hierarchically decomposes system functions into objects, methods, and operations. The psychological hypothesis underlying the CLG is that ". . . to design the user interface is to design the user's model" (Moran, 1980, p. 296). Idealized user knowledge is represented with a somewhat complex grammar having the appearance of a high level programming language. System structure has three components—conceptual, communication and physical—and two different levels within each component. The assumption is that users form a layered mental representation of systems, and these components and levels are intended to mirror the layers of representation. Essentially, the CLG is a design tool to separate the conceptual model of a system from its specific command language and to reveal the relationship between the two ((Wilson, Barnard, Green, & Maclean, 1988., p. 53), quoting Moran (1981, p. 5)). The CLG formalism offers a thorough and broad framework for describing many aspects of a user interface. Like GOMS, however, creating a CLG description is complicated and time-consuming. While the CLG concept has received much discussion and acceptance in human-computer interaction literature over the past decade, there seems to be some doubt about its utility in evaluation of real interaction designs (Wilson, et al., 1988).

The keystroke-level model ((Card, et al., 1980), as discussed in Reisner, (1983)), models user-system interaction at—as the name implies—the level of keystrokes, and also includes other simple physical actions (e.g., mouse clicks) at that same level of granularity. A model of the user, derived from a structural representation of a command language system, is intended as a system design tool to predict expert user task time for routine tasks (Reisner, 1983, p. 21). A method for performing a task is specified as a series of commands. Predictions of user performance are made by counting keystrokes and other similar actions such as pointing (e.g., cursor movement), homing (e.g., moving hands to the keyboard), and simple drawing actions. Another parameter includes mental preparation (e.g., deciding to make a particular keystroke). Like GOMS and the CLG, the level of detail in a keystroke analysis can be overwhelming.

The Reisner Action Language Model (1981) provides a technique for comparing predicted user performance associated with alternative designs, and for identifying design decisions that might cause user errors. Physical actions of the user are viewed as expressions in the language of the user. Complexity measures are applied to formal grammars of the user's language, revealing inconsistencies (e.g., more grammar rules are required for exceptions to a consistent description). Reisner applied this model to the Robart graphics system interface, describing its interface in the Action Language and then applying metrics to predict user performance to make comparisons of alternative designs and to identify design choices that could cause users to make mistakes. Like the CLG, the Action Language models computer and user as two cooperating and communicating information processors.

The Task Action Grammar (TAG) (Payne, et al., 1986) is another formal user model—specifically, a cognitive competence model—with a command language orientation. A metalanguage of production rules encodes generative grammars that convert simple tasks into action specifications. As in Reisner's work, a goal of TAG is to capture the notion of consistency. Marking of tokens in production rules with semantic features of the task allows representation of family resemblances, a way of capturing generalities of which the user may be aware (Wilson, et al., 1988, p. 58). Complexity measures taken on the production rules are predictors of learnability.

More recent work in cognitive modeling for analytic evaluation has largely been in the area of extensions to the original GOMS-related approaches (e.g., (John, 1990)), combining GOMS-like models with other cognitive models for problem-solving. Some examples include Soar, a theory of cognition that provides an integrated architecture for exploring different user behaviors (Peck & John, 1992); and other areas such as documentation (Gong & Elkerton, 1990) and graphic machine-paced tasks (John & Vera, 1992).

3.2. Usability Inspection Methods

Researchers have recently begun exploring new approaches to user interface design evaluation, seeking ways to increase the efficiency of the process. A goal of their work is to find techniques for identifying at least major usability design errors without the cost and time consumption of empirical testing, specifically to find methods for usability evaluation that are more accessible to system developers and that will, accordingly, be used more often in real world development projects. These techniques are based on *inspection* of interfaces, with core tasks being inspected by expert evaluators typically using design representations or early interface prototypes. We include them in this section on analytic evaluation, because the essence of their approach is analysis.

One type of usability inspection method includes interface design *walkthroughs*, an evaluation concept adapted from software engineering and other design-oriented disciplines. A well-known type of these walkthrough techniques is the cognitive walkthrough (Lewis, Polson, Wharton, & Rieman, 1990), a theory-based evaluation technique focusing on an evaluator's expectations of a user's cognitive activities. The design is inspected, through a structured list of questions, for a match to users' goals and knowledge. Forms are used to guide the evaluator through the steps of the process. Success of the technique has been variable, depending on the cognitive science knowledge and skills of evaluation team members (Wharton, Bradford, Jeffries, & Franzke, 1992).

Nielsen's work on heuristic evaluation (Nielsen & Molich, 1990) is another example of an inspection method. Here the focus is on design guidelines rather than cognitive activities. Usability of an interface design is judged informally, based on intuition and opinions of evaluators, against heuristics that are a small set of general usability guidelines. While individual evaluators using this approach did not tend to find large percentages of existing usability flaws, aggregating over a set of several independent evaluators effectively solved this problem. This approach was validated by separate empirical studies (Nielsen, 1992) in which he found that non-usability-specialists were not particularly effective at uncovering usability design problems, usability experts were better, and experts in both usability and the application domain performed best. Heuristic evaluation, along with scenarios (partial prototypes (Nielsen, 1987)) and small thinking aloud studies (Nielsen, 1988), constitute what Nielsen calls "usability engineering at a discount" (1989). The advantages he states for this general approach are low cost, intuitiveness, ease of motivating developers to do it, and effectiveness early in the development process.

Walkthrough methods generally fare well in comparisons with conventional empirical usability evaluation (Jeffries, Miller, Wharton, & Uyeda, 1991; Karat, Campbell, & Fiegel, 1992; Lewis, et al., 1990). However, there are some contradictions among results, and we are warned (Jeffries & Desurvire, 1992) that such results are not to be taken as indications that inspection methods can replace empirical evaluation; there are many additional qualitative benefits from usability testing with users. In fact, Lewis et al. (1990) explicitly state they do not wish "to claim that the cognitive walkthrough methodology will eliminate the need for evaluating prototypes of the interface." Their finding, based on empirical studies, is that cognitive walkthroughs are inexpensive (about one hour per task that is inspected), yet detect nearly half the usability problems found by empirical evaluation with users of a design. In contrast, Karat et al. (1992) found that empirical testing required the same or less time to identify each usability problem when compared to walkthrough methods.

Another example of a usability inspection method is claims analysis, a method developed by Carroll, Rosson, and Kellogg within their task-artifact framework (Carroll, 1990; Carroll, Kellogg, & Rosson, 1991). The task-artifact framework is based on a view of the world that contains user tasks, which are the context for artifacts (interface objects that interface developers build). In order to bring theory to bear on the interaction design activity, they propose the task-artifact cycle to formalize and speed up the connection between theoretical analysis and design. Their underlying hypothesis is that interface artifacts represent theories of the designer, containing claims about potential users and how they use the interface. The task-artifact cycle is a kind of instance of the classical epistemological cycle of theory formulation and observational validation. The design rationale is inferred from the design and stated in terms of cause-and-effect relationships between artifacts and psychological consequences of their use (Carroll, et al., 1992, p. 4).

3.3. A Brief Critique of Analytic Evaluation

Since the focus of this paper is not analytic evaluation, we will not give a detailed critique of the analytic approach to user interface evaluation. However, we will list some of the more common comments encountered in the relevant literature. For further reading, we suggest, for example, Carroll and Rosson (1985). General criticism of analytic evaluation methods, which we have gleaned from a variety of sources and our own observations, revolves around the following issues:

- Because these methods support analysis but not synthesis, they are not directly supportive of interaction design. To improve an existing interaction design, a developer must try some (random, unless supported in some other way) new design idea, build a large representation model, conduct the analysis, and see if the predicted performance is better.

- Empirical evaluation gives much more useful data—including qualitative data that can help identify specific problems in an interaction design—in terms of directly affecting usability.

- Almost all analytic approaches assume error-free, expert task performance. But error-free performance is obviously is not the typical experience of a user of an interactive system. And it is, indeed, study of error-related situations that often reveals the greatest indicators of usability problems.

- While most of the analytic approaches mentioned above produce a representation that includes a task structure, none deal with the temporal relations necessary to provide a complete user-task-oriented representation of a design.

- Most analytic approaches require that the design be complete and a full global model built for the entire interface before analysis, at any level, can proceed. An approach supporting local analysis—analysis of one area of design problems independent of the overall design—would be more cost effective.

- Many of these approaches are based on an open-loop model of command sequences wherein the user enters a command, then the system executes it. The model we seek is a model of highly interactive direct manipulation, involving incremental user actions concurrent with perception and reaction in a closed-loop feedback cycle.

While analytic evaluation of user interfaces has been, and continues to be, an open area of human-computer interaction research, we find that the sorts of issues just presented limit the efficacy of many of these analytic evaluation techniques. Because of these limitations, we do not foresee analytic formative evaluation replacing empirical formative evaluation.

However, in all fairness, GOMS-based analytic evaluation has been shown to be effective for real world problems within specific classes of tasks (e.g., constrained and repetitive tasks such as those of a telephone assistance operator). In such an application, CPM-GOMS, which is GOMS analysis combined with the critical path method of project management, was shown to be remarkably accurate in predicting performance, allowing analysts to sort out the complexity of parallel operator activities (e.g., (Gray, John, & Atwood, 1992; Gray, John, Stuart, Lawrence, & Atwood, 1990)). In this method, cognitive, perceptual, verbal, and motor actions for a task, plus response time of the system, are represented on a critical path timing chart. This supports determination of the effect on user performance of a given change in task procedures or in keyboard layout. This method supports prediction of the effect of, for example, adding voice recognition to the system, or changing system response time. 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 usefulness of GOMS, at least in this application domain, was indeed validated by a series of studies (Gray, et al., 1992). Olson and Olson (1990) give a good review of the current status of GOMS-related analytic modeling.

4. EMPIRICAL FORMATIVE EVALUATION

Now we focus our discussion to the class of formative evaluation techniques based directly on empirical, observational data from user testing without a strong component of modeling or analysis (except for analysis of collected observational data, of course). Much of what we will say about empirically based formative evaluation is now part of what is generally called *usability engineering*, coming from the work of, for example, (Good, Spine, Whiteside, & George, 1986; Nielsen, 1989; Whiteside, Bennett, & Holtzblatt, 1988).

4.1. Types of Empirical Formative Evaluation Data

There are several types of data that are produced during empirical formative evaluation, to be used in making decisions about iterative redesign of the user interface. These types of data include:

- *Objective* — These are directly observed measures, typically of user performance while using the interface to perform benchmark tasks.
- *Subjective* — These represent opinions, usually of the user, concerning usability of the interface.
- *Quantitative* — These are numeric data and results, such as user performance metrics or opinion ratings. This kind of data is key in helping monitor convergence toward usability goals during all cycles of iterative development.
- *Qualitative* — These are non-numeric data and results, such as lists of problems the user had while using the interface. This kind of data is useful in identifying

which design features are associated with measured usability problems during all cycles of iterative development.

Even though objective evaluation is often associated with quantitative data and subjective evaluation with qualitative data, subjective evaluation (e.g., using user preference scales or questionnaires) can also produce quantitative data. Also, objective evaluation activities (e.g., benchmark task measurements) can produce qualitative data (e.g., critical incidents and verbal protocol, discussed in Section 4.4.2 on Qualitative Data Generation Techniques, below).

4.2. Steps in Empirical Formative Evaluation

There are several major steps in empirical formative evaluation, including:

- Developing and conducting the experiment
- Generating and collecting the data
- Analyzing the data
- Drawing conclusions
- Redesigning and implementing the revised interface

We do not intend, in this paper, to give details on each step; we will present an overview to give the reader an appreciation for developing an experiment that allows collection of data that can be analyzed to determine usability of the user interface—the formative evaluation process. In-depth coverage of each of these steps, including hands-on exercises and solutions, can be found in (Hix & Hartson, 1993).

While many members of an interface development team may be involved in performing any or all of these steps at various times, we will refer to the person who is primarily responsible for these activities as the user interaction design evaluator, or simply the *evaluator*.

4.3. Developing and Conducting the Experiment

Developing and conducting an experiment to be used for formative evaluation involves three main activities, not necessarily strictly in the order given:

- Selecting participants to perform tasks
- Developing usability goals and experimental tasks
- Directing an experimental session

4.3.1. Selecting Participants

Participant selection involves determining appropriate users for experimental sessions. ("Participant" is the term that most recent human factors literature now uses to indicate a human subject taking part in an experiment.) The evaluator must determine the classes of *representative users* that will be used as participants to evaluate the interface. This kind of user should be somewhat knowledgeable of the interactive system application domain (e.g., word processor, spreadsheet, graphics drawing application, process control system, airline reservation system, or whatever), but not necessarily knowledgeable of a specific interactive system within that domain. These participants should represent typical expected users of the interface being evaluated, including their general background, skill level, computer knowledge, application knowledge, and so on.

In addition to representative users, the *human-computer interaction expert* plays an important part in formative evaluation. Evaluators sometimes overlook the need for critical review of the interface by a human-computer interaction expert when developing a formative evaluation plan. Such an expert is broadly knowledgeable in the area of interface development, has extensively used a wide variety of interfaces, knows a great deal about interaction design and critiquing, and is very familiar with interaction design guidelines. An exper will find subtle problems that a representative user would be less likely to find (e.g., small inconsistencies, poor use of color, confusing navigation, and so on). More importantly, a human-computer interaction expert will offer alternative suggestions for fixing problems, unlike the representative user who typically tends to find a problem but may not be able to offer suggestions for solving it. An expert can draw on knowledge of guidelines, design and critiquing experience, and familiarity with a broad spectrum of interfaces to offer one or more feasible, guideline-based suggestions for modifications to improve usability.

The number of participants needed for each cycle of formative evaluation is surprisingly small. Each cycle needs a few carefully chosen representative users, and one or maybe two human-computer interaction experts. In fact, the purpose of formative evaluation is not to focus on a large number of experiments with a large number of participants for each one. Rather it is to *focus on extracting as much information as possible from every participant* who uses any part of the interface (Carroll, et al., 1985; Whiteside, et al., 1988). Some empirical work (Nielsen, et al., 1990) has shown that the optimum number of participants for a cycle of formative evaluation is three to five per user class. Only one participant per user class is typically not enough. But more than ten participants per class is not worth the diminishing returns obtained; after about five or six participants, they tend to cease finding new problems, and mostly reiterate the ones already uncovered by prior participants. Often three participants per well-defined user class is the most cost effective number. Empirical work by Virzi (1992) corroborates these suggestions, and, in fact, found that 80 percent of usability problems are discovered with four or five participants, and that additional participants are less and less likely to uncover new and/or severe problems.

4.3.2. Developing Usability Goals and Experimental Tasks

We have said that a key to determining usability in an interaction design is quantifiable usability goals. These are operationally defined criteria for assessing usability. Typically called *usability specifications*, they are established by the development team, and are the comparison point for actual observed user performance.

Usability specification tables are a convenient way to record established usability metrics. The concept of formal attribute specification in tabular form, with various metrics operationally defining success, was developed by Gilb (1981). The focus of Gilb's work was use of measurements in managing software development resources. Bennett (1984) adapted this approach to usability specifications as a technique for setting planned levels and managing development to meet those levels. These ideas were refined and integrated into the usability engineering concept in (Good, et al., 1986) and (Whiteside, et al., 1988). *Usability engineering*, as defined in (Good, et al., 1986), is a process through which usability characteristics are specified, quantitatively and early in the development process, and measured throughout the process. Carroll and Rosson (1985) also stressed the need for quantifiable usability specifications, associated with appropriate benchmark tasks, in iterative development of user interfaces. Without measurable specifications, it is impossible to determine either the usability goals of a product or whether the final product meets those goals.

We have, through years of working with real world developers and from our own evaluations, adapted the format in (Whiteside, et al., 1988), revising it into the form shown in Figure 3. The example used in this usability specification table is a graphical drawing application on which we performed a complete formative evaluation for a client.

Usability Attribute	Measuring Instrument	Value to be Measured	Worst Acceptable Level	Planned Target Level	Best Possible Level	Current Level
Initial use	"Create a line drawing" benchmark task	Length of time to create line drawing on first trial	30 secs.	15 secs.	5 secs.	10 secs.
First impression	Questionnaire	Average rating (range -2 to 2)	0	0.5	1.25	1.0

where:

usability attribute is the usability characteristic to be measured; some common attributes include initial use, learnability, retainability, first impression, long-term user satisfaction, and so on

measuring instrument is a description of the method for providing a value; for example a benchmark task is an objective instrument and a questionnaire is a subjective instrument

value to be measured is the metric for which data values are collected (using the measuring instrument) during an evaluation session with a user; for example, length of time or number of errors in performing a benchmark task, or a rating on a questionnaire

worst acceptable level is the lowest acceptable level of performance for the attribute; the border of failure for usability

planned target level is the target indicating attainment of unquestioned usability success for the current version; the "what we would like" level

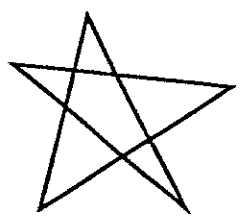
best possible level is a realistic state-of-the-art upper limit; the inspiration level

current level is the currently known level of the value to be measured for the attribute (when available)

Figure 3. Examples of usability specifications for a graphical drawing application

Let us briefly explain the usability specifications shown in Figure 3. An important usability attribute for virtually any interface is how quickly users can perform tasks the first (initial) time they use it. Thus, in the first attribute, user performance during initial use of the interface for the specific benchmark task to "create a line drawing" will be measured. This is a common task that users of such a system would probably want to perform frequently, quickly, and easily.

For each benchmark task cited in a usability specification table, a specifically worded benchmark task must be written telling the user what to do (but not how) during an evaluation session. For example, for the "create a line drawing" benchmark, the task description might be: Produce a line drawing similar to the approximate size and shape of this sketch:



As shown in Figure 3 for this simple example, we state that initial use time to perform the task to "create a line drawing" must be, at the longest (worst level), 30 seconds. The expected (planned) length of time to perform this task the first time is 15 seconds, while the very best time to perform it is estimated at 5 seconds. A similar example, for the usability goal desired for "first impression" of the graphics application, is also shown.* The first usability attribute is associated with collection of objective data, specifically by measuring performance of the user on a particular benchmark task. The second usability attribute is associated with collection of subjective data, measuring the user's opinion of the interface based on a questionnaire. Both measures are quantitative.

In addition to benchmark tasks developed for the usability attributes, the evaluator may also identify other *representative tasks* for participants to perform. These tasks will not be tested quantitatively (that is, against usability specifications) but are deemed, for whatever reason, to be important in adding breadth to evaluation of the user interaction design. These additional tasks, especially in early cycles of evaluation, should be ones that users are expected to perform often, and therefore should be easy for the user to accomplish. In early cycles of evaluation, these representative tasks, together with the benchmark tasks, might constitute a core set of tasks for the system being evaluated, without which a user cannot perform useful work. All task descriptions should, in general, be written down rather specifically and should state what the user should do, rather than how the user should do it.

In addition to strictly specified benchmark and representative tasks, the evaluator will usually find it useful to observe the user in informal *free use* of the interface, without the constraints of predefined tasks. Benchmark tasks, other representative tasks, and free use are all key sources of critical incidents (see Section 4.4 on Generating and Collecting the Data, below), a major form of the qualitative data to be collected. Free use by the participant is usually performed after some or all predefined tasks have been completed, especially those related to the initial use attribute. To engage a participant in free use, the evaluator might simply say to the participant, "Play around with the interface for awhile, doing anything you would like to, and talk aloud while you are working." We will discuss verbal protocol taking during an evaluation session in Section 4.4 on Generating and Collecting the Data, below. Free use is valuable for revealing user and system behavior in situations not anticipated by designers—often situations that can "break" a poor design.

* There are heuristics for determining usability attributes and values of the various levels (Whiteside, Bennett, & Holtzblatt, 1988); however, we will not present a tutorial on establishing usability specifications in this paper.

4.3.3. Directing an Experimental Session

In order to properly direct an experimental session, the evaluator must produce training materials and determine experimental procedures—exactly what will happen during a test session with a participant. The evaluator must decide on whether laboratory testing or field testing, or both, will be performed. *Laboratory testing* involves bringing the user to the interface; that is, users are brought into a usability lab setting where they perform the benchmark tasks, performance measures are taken as appropriate, free use is encouraged, and so on. *Field testing* involves bringing the interface to the user; that is, the current version is set up in situ, in the normal working environment in which the user is expected to use the interface, and more qualitative, longer-term data are often collected.

Obviously lab and field testing each have pros and cons. In a laboratory setting, the evaluator can have greater control over the experiment, but the conditions are mostly artificial. On the other hand, in a field test, the evaluator has less control, yet the situation is more realistic. Laboratory testing is typically more appropriate for earlier cycles of formative evaluation, when major problems with an interaction design are typically discovered. Field testing works well for later cycles, when data on longer-term performance with the interface may be desirable. A combination of the two is the ideal circumstance for formative evaluation, but, in real life, true field testing may be limited or even impossible. In this case, laboratory testing may have to suffice.

In conjunction with developing experimental procedures, the evaluator should prepare *introductory instructional remarks* that will be given uniformly to each participant. These remarks should briefly explain the purpose of the experiment, tell a little bit about the interface the participant will be using, state what the user will be expected to do, and the procedure to be followed by the user. It is also important to make very clear to the participants that *the purpose of the session is to evaluate the system, not to evaluate them.*

Finally, an *informed consent form* must be prepared for each participant to sign. This form states that the participant is volunteering to participate in the experiment, that the data may be used if the participant's name or identity is in no way associated with the data, that the participant understands the experiment is in no way harmful, that the participant may discontinue the experiment at any time, and so on. This is standard protocol for performing experiments using human participants, and is to protect both the evaluator and the participant. The informed consent form is legally and ethically required; it is not optional.

When benchmark tasks have been developed, the setting and procedures have been determined, and the types of participants chosen, the evaluator must perform some *pilot testing* to ensure that all parts of the experiment are ready. The evaluator must make sure that all necessary equipment is available, installed, and working properly, whether it be in the laboratory or in the field. The experimental tasks should be completely run through at least once, using the intended hardware and software platform (e.g., the interface prototype) by someone other than the person(s) who developed the tasks, to make sure, for example, that the platform supports all the necessary user actions and that the task instructions are unambiguously worded.

During an evaluation session, the evaluator gives the participant appropriate instructions, has them sign the informed consent form, and administers the tasks. When the participant has performed the desired tasks, including completion of any questionnaire or survey, it is common practice for the evaluator to give the participant some sort of "reward" (e.g., money, mug, t-shirt, cookies). While it is often necessary to offer compensation in order to recruit participants, some practitioners believe monetary rewards can bias results. For example, it is possible that paid participants with greater financial need could be more motivated than participants without a financial need to perform for pay in a study. A possible misconception by a participant could be that good performance will lead to approval and therefore more "employment."

4.4. Generating and Collecting the Data

We have already mentioned qualitative and quantitative data. There are methods for generating and collecting both kinds, discussed in the following sections.

4.4.1. Quantitative Data Generation Techniques

Quantitative techniques are used to directly measure observed usability levels in order to compare against usability specifications. There are two kinds of quantitative data generation techniques most often used in formative evaluation, namely:

- Benchmark tasks, and
- Questionnaires.

We have already discussed development of *benchmark tasks* (in Section 4.3.2, above). During the experiment each participant performs the prescribed benchmark tasks, and the evaluator takes numeric data, depending on what is being measured. For example, the evaluator may measure the time it takes the participant to perform a task, or count the number of errors a participant makes while performing a task, or count the number of tasks a participant can perform within a given time period.

The second quantitative data generation technique is *questionnaires,* or user preference scales, for different features that are relevant to usability of the interface being evaluated. Thi kind of questionnaire or survey is inexpensive to administer, but is not easy to produce so that it is valid and reliable. It is are the most effective technique for producing quantitative data on subjective user opinion of an interface. The Questionnaire for User Interface Satisfaction, or QUIS, survey (Chin, Diehl, & Norman, 1988) is the best of these validated questionnaires.

4.4.2. Qualitative Data Generation Techniques

Qualitative data are extremely important in formative evaluation of a user interaction design. The kinds of techniques that are most effective for generating qualitative data include the following:

- Verbal protocol taking,
- Critical incident taking, and
- Structured interviews.

Perhaps the most common technique for qualitative data generation is *verbal protocol taking*, also called "thinking aloud." Here, the evaluator asks a participant to talk out loud while working, indicating what they are trying to do, or why they are having a problem, what they expected to happen that didn't, what they wished had happened, and so on. This technique obviously is invasive to the participant, so it should be used sparingly with benchmark tasks where timing is important. But it is immensely effective in determining what problems a participant is having and what might be done to fix those problems. The verbal protocol technique is best employed during free use of the system or during other non-timed predefined task performance by a participant. The evaluator will find that some participants are not good at thinking aloud while they work; they will not talk much and the evaluator will constantly have to prod them to find out what they are thinking or trying to do. It is perfectly acceptable for the evaluator to query and prompt such reticent talkers, in order to produce the desired information. Remember, one of our goals in formative evaluation is not to have a large number of participants, but rather to extract as much data as possible from each and every participant. Evaluators become more skilled at this as they work with more participants. The key to effective prompting is to give the participant helpful hints (e.g., "Do you remember how you did this before?", "What do you think the so-and-so menu is for?", "What did you expect to happen then?", and so on), but to refrain from telling them exactly what to do.

Another kind of qualitative data generation that works well, often in conjunction with verbal protocol taking, is *critical incident taking*. A critical incident is something that happens while the participant is working that has a significant effect, either positive or negative, on task performance or user satisfaction, and thus on usability of the interface. A bad, or negative, critical incident is typically a problem the participant encounters—something that causes an error, something that blocks (even temporarily) progress in task performance, or something that results in a pejorative remark by the participant. For example, an evaluator might observe a participant try unsuccessfully five times to enlarge a graphical image on the screen using a graphical editor. If it is taking the participant so many tries to perform the task, it is an indication that this particular part of the design should be improved. Similarly, the user may begin to show signs of frustration, either with remarks or actions.

An occurrence that causes the user to express satisfaction or closure in some way (e.g., "That was neat!", "Oh, now I see", "Cool!", and so on) is a good, or positive, critical incident. When a first-time user immediately understands the metaphor of how to manipulate a graphical object, that can also be a positive critical incident. While negative critical incidents indicate problems in the interaction design, positive critical incidents indicate metaphors and details that, because they work well or participants like them, should be considered for use in other appropriate places throughout an interface. Critical incidents can be observed during performance of benchmark tasks, other representative tasks, or when a participant is freely using the system.

Structured interviews provide another form of qualitative data. These are typically in the form of a post-experiment interview, a series of preplanned questions that the evaluator asks each participant. Such a post-session interview might include, for example, such general questions as "What did you like best about the interface?", "What did you like least?", "How would you change so-and-so?", and so on.

4.4.3. Data Collection Techniques

There are several recommended techniques for capturing both qualitative and quantitative data from participants during a formative evaluation experimental session, including:

- Real-time notetaking,
- Videotaping,
- Audiotaping, and
- Internal instrumentation of the interface.

We have found, through our own experience and numerous conversations with other evaluators, that *notetaking in real time* is still the most effective technique to use for data capture during a usability evaluation session. The evaluator should be prepared to take copious notes, either with pencil and paper or using a word processor, as activities proceed during the session. Because a multitude of simultaneous activities can happen fast in a session, when an evaluator is directing a test session for the first few times, it is a good idea to have a second evaluator observe the session in order to help take notes.

To collect quantitative data, the required equipment is minimal: a stop watch for timing participants performing tasks, and some kind of tally sheet for noting and/or counting errors. To collect qualitative data, the evaluator(s) should write down all observed critical incidents, as well as any other observations, as a participant performs each task or uses the interface freely.

Videotaping is a well-known and frequently-used data collection technique. Videotaping has many advantages, including detailed capture of what occurs during an experiment. A camera aimed at the participant's hands and the screen is the most important, and a second, if available, should be aimed for a broader view, including the participant's face.

However, the problem with analysis of videotape is two-fold. First, it can take as much as eight hours to analyze each one hour of videotape (Mackay & Davenport, 1989). The chances of laboriously going back through several hours of videotape from half a dozen evaluation sessions is therefore very slim. Second, with multiple views and/or tapes of the same test session, there is a problem of synchronization of the tapes. The main use of videotape should be as a *backup* for what happened during a test session, not as the main source of data to be captured and analyzed. For example, in case of confusion, uncertainty about a specific detail, or some missed part of a critical incident that occurred during an evaluation session, the evaluator can go to a specific point on the videotape and review a very short sequence to collect the missing data.

We have also found that a few carefully selected videoclips (say, of five minutes each or less) can be of great influence on a design team that is resistant to making changes to what they believe to be their already perfect design. We have seen programmers who had the major responsibility for an interaction design watch videoclips in awe while a bewildered participant struggled to perform a task with an awkward interface. These same clips are also useful in convincing management that there is a usability problem in the first place.

Finally, *internally instrumenting the interface* being evaluated is a useful way to capture the kinds of data we have been discussing. For example, data on user errors or frequency of command usage, or automatically computing elapsed task times from start/stop times, can be automatically gathered by a fairly simple program. There is, however, a potential problem with this technique. Evaluators may think "the more data the better" but find themselves inundated with details of keystrokes and mouse clicks. A fairly short session can produce a several megabyte user session transcript file. Manual analysis of a file dump is totally untenable. But the difficult question is: what analysis should be done once such data are extracted from a transcript file? How can, for example, any of these keystrokes or cursor movements be associated with anything significant—either good or bad—happening to the user, and therefore related to usability? The only feasible way in which such data might be useful is if their analysis can be automated, and we know of very few viable techniques for analyzing (either manually or automatedly) user session transcripts. We will briefly discuss tools, including those for analysis of user session transcripts, in Section 6 on Future Trends, below.

4.5. Analyzing the Data

After all evaluation sessions for a particular cycle of formative evaluation are completed, the data collected during those sessions must be analyzed. We will not, in general, be performing inferential statistical analyses, such as analyses of variance (ANOVAs) or t-tests or F-tests. Rather, we will be using some data analysis techniques to help us determine if we have met our usability specification levels, and if we have not, how to modify the design to help us converge toward those goals in subsequent cycles of formative evaluation.

The first step in analyzing the data is to compute averages and any other values stated in the usability specifications (timing, error counts, questionnaire ratings, and so on). The evaluator can then enter a summary of the results into an Observed Results column added on to the usability specification table. By directly comparing observed results with the specified usability goals, the evaluator can tell immediately which usability specifications have been met and which have not been met during this cycle of formative evaluation. If all worst acceptable levels have been met and enough planned target levels have been met to satisfy the development team that usability of the current version of the design is acceptable, then the design is satisfactory and iteration for this version can stop. The one exception is if, for whatever reason, there is suspicion that the usability specifications may be too lenient, and are not therefore a good indicator of high usability. Then, obviously, the development team should reassess the usability specifications to see if they should be more (or less) stringent.

If usability specifications have not been met (the most likely situation after the first cycle of testing), then more in-depth data analysis should be performed. Our goal in further data analysis—much of which will be qualitative data analysis—is structured identification of the observed problems and potential solutions to them. We will then address solving those problems in order of their potential impact on usability of the interface. The process of determining how to convert the collected data into scheduled design and implementation solutions is essentially one of negotiation in which, at various times, all members of the development team are involved.

Figure 4 shows a form that we have found to be useful in enumerating and organizing the multitude of problems that will inevitably be uncovered during a cycle of formative evaluation. We will use some data from our own formative evaluation of the graphical drawing application mentioned earlier, as an example to explain each column in this table.

Problem	Effect on User Performance	Importance	Solution(s)	Cost	Resolution
Too much window manipulation	10 of 35 minutes	high	fix window placement automatically but allow user to reposition it	6 hrs.	
Black arrow on black background	?	low	reverse arrow to white on black	1 hr.	

where:

problem is an interface problem observed as users interact with the system during evaluation; usually identified from (negative) critical incidents

effect on performance is data about the amount of time spent by the user dealing with a specific problem

importance is a subjective indication, produced by the development team, of a problem's overall effect on user performance and interface usability; generally rated as high, medium, or low

solution(s) is one or more proposed design changes to solve a problem

cost is the resources—usually time and/or money—needed for each proposed solution

resolution is the final decision made to address each problem

Figure 4. Example of cost/importance table for organizing and analyzing observed problems

Let us briefly explain the first interface problem shown in Figure 4 for the graphical drawing application. In this application, whenever a new window appeared, it remained connected to the cursor, and therefore wandered around on the screen following mouse movement, until the user clicked a mouse button to purposefully place the window. The evaluator, during evaluation of several tasks, observed that users intensely disliked this design feature, because it distracted them from whatever task they were trying to do when a "wandering window" appeared, stuck to the cursor.

After all evaluation sessions, the evaluator makes a complete list of observed *problems*—potentially hundreds of them—and, as with usability specifications, a team decides on appropriate values for the other columns for each problem listed. In the graphical drawing application interface, after about four hours of the first cycle of evaluation sessions, we had a list of 54 problems to tackle, ranging from serious to trivial. Two of those problems are shown in Figure 4. As we have already mentioned, the earliest cycles of formative evaluation typically give the most data and therefore result in the longest lists of problems. Later cycles generally produce fewer changes, especially if the process is leading to convergence toward a more usable design.

The *importance* rating results from a dialectic decision-making process among the entire development team (possibly including some users), and can take into account many factors, including impact on overall system integrity and consistency, intuition, and judgement about pragmatic development concerns. First candidates for high importance ratings are those that, as revealed by impact analysis (see below), have the greatest effect on meeting the established usability specifications. The wandering windows was definitely of high importance, because of the constantly mounting irritation it caused users and the amount of time it distracted them from their tasks.

The development team must also propose one or more possible *solutions*, that is, changes to the design to solve each of the observed problems on the list. Ideas for these proposed design solutions can come from a variety of places, including design principles and guidelines, suggestions by participants, known available technology, and study of other similar designs. More than one possible solution is often appropriate, and different solutions may have very different costs associated with them. That is, implementing one solution to a problem may be estimated to take two hours, while a different solution to the same problem may be estimated to take two days of coding time. When, in a later step of data analysis, decisions are made about which problems are most critical to fix, it may be useful to have alternative solutions with different costs, so that at least some sort of solution can be offered for a problem that might otherwise have to be ignored because the cost of the first choice solution is too high. For our wandering windows problem, the solution was to fix window placement automatically at the center of the screen when a window first appears, but allow the user to move and reposition it anywhere on the screen at any time. We then retested this redesign in future formative evaluation cycles.

For each proposed solution, a *cost* of implementing it must be estimated. This cost is usually the amount of resources, typically time (sometimes money) needed for making the indicated change. Typically it is measured in terms of the number of hours needed to modify a prototype, or to modify existing source code or to write new code. To fix the placement of a window, the programmer on the development team estimated that six hours of recoding would be necessary. Lower costs here are typically obtained when changes are made to a prototype, rather than to a version of the interface that has been coded. This reinforces the need to select good prototyping tools and to get a prototype running and testable as early in the development process as possible.

Finally, after cost/importance analysis and/or impact analysis (see below) of all problems in the list, a *resolution*—a final decision—is made for each problem in the list. This is an indication of how each problem will be addressed (e.g., "do it"; "do it, time permitting"; "postpone indefinitely") and which solutions will be implemented. After our general analysis of the graphical drawing application, we decided that the wandering windows had to be modified, along with more than a dozen other serious problems.

When choosing which problems have the highest priority for changes, high importance problems are addressed first, while lower importance problems receive later attention, resources permitting. Ideal, of course, are high importance/low cost problems (e.g., confusing error messages). This approach gives much more controllability and accountability over redesign and iteration than an ad hoc approach in which evaluators and others involved in development of the user interface simply make some guesses about which identified problems to address and in which order.

An example of another problem we encountered in an evaluation session of the graphical drawing application, the second problem listed in Figure 4, was simply an oversight on the part of the designer. The cursor icon was a black arrow in one particular mode of the application. However, when the black arrow was moved over a black background, it stayed black and therefore disappeared. Although this was annoying, users spent very little time in this problem and it did not measurably interfere with users' performance. Its importance was therefore low. Our proposed solution was obvious: to reverse the arrow to white when on a black background, and the programmer estimated this would take about one hour. Our resolution, after our full analysis, was to do it. The basis for the decision was really more related to fixing a simple, but silly, oversight on the part of a designer, rather than modifying a problem that had a big impact on usability.

Once all columns except the Resolution column have been completed for all observed problems, we can perform a couple of different kinds of analyses that help determine a resolution for each problem, focusing on which changes will have the greatest impact on usability. These analyses include:

- Cost/importance analysis, and
- Impact analysis.

In a *cost/importance analysis*, we consider the relative costs and importances of the problems as listed in a table like the one in Figure 4. We must first determine the resources (in particular, time and people) we have available to allocate for making modifications. In our evaluation of the graphical drawing application, we had 60 hours (across two programmers) allocated for modifications in our first cycle of formative evaluation. We totalled the Cost column to find we had more than 140 hours of suggested changes, even after selecting the lowest-cost suggestions to change for those problems for which we had multiple suggestions. We had some difficult decisions to make about which problems were going to get addressed, and which were going to remain unchanged, at least for the next cycle of evaluation.

As already mentioned, another kind of analysis, called *impact analysis* (e.g., (Good, et al., 1986)), can be used to determine what problems most affect achievement of the usability specifications we have previously defined. This analysis is based on the time a user spends in various problems, recorded in the Effect on User Performance column; this time is often the biggest contributor to not achieving the desired usability specifications. Comparing usability specification levels, observed values, and values from the Effect on User Performance column directly indicates which problems have the largest effect on meeting the usability specifications. Because the effect on user performance time is deducted from user performance times when the associated problem is solved, this will have the most impact on meeting the usability specifications. Because of this direct effect, a high importance rating is generally assigned to problems with the greatest impact on performance. In the wandering windows example, a quick scan of the videotaped sessions showed that one user spent 35 minutes attempting to perform several benchmark tasks, of which about 10 of those 35 minutes were wasted with window placement (see second column of Figure 4). This kind of data provides a good predictor of the effect of an interface problem on achieving a usability specification. For this situation, absence of the problem would have reduced user performance time to 25 minutes across tasks.

To compare the various kinds of data collection and analysis techniques we have discussed, we can classify them into the simple taxonomy shown in Figure 5.

	Quantitative (numeric)	*Qualitative (non-numeric)*
Objective (measures performance)	•Impact analysis •User performance metric (benchmarking)	
Subjective (involves opinion)	•Cost/importance analysis •User satisfaction metric (user preference)	•Critical incident analysis •Protocol taking/analysis •Structured interviews

Figure 5. A simple taxonomy of data collection and analysis techniques

A summary of the ways in which the various kinds of data—quantitative and qualitative—are primarily used in formative evaluation is shown in Figure 6.

However, there is considerable overlap among these categories. For example, participants themselves can often suggest solutions for usability problems encountered during evaluation. Also, quantitative data from satisfaction questionnaires may reveal not only that something is wrong, but give a strong indication of what is wrong.

USE THESE:	*TO DECIDE THESE:*
Quantitative data	Something is wrong
Qualitative data	What is wrong
Designers	How to fix what is wrong

Figure 6. Comparison of uses for quantitative data vs. qualitative data in formative evaluation

4.6. Drawing Conclusions

We can finally now complete the Resolution column of Figure 4. If the list is ordered by importance (descending, high to low) and, within that, cost (ascending, low to high), with high importance/low cost at the top of the list followed by high importance and moderate/high cost and so on, we can determine how many problems can be addressed given the time and other resources allotted for modifications. Problems at the top of the list get first priority, unless, for example, some of the high importance/high cost problems are so critical that they must be fixed despite their high cost.

For our graphical drawing application, there were 24 different problems with high importance and low/moderate/high cost. Some of the high importance/high cost problems had to be included to make parts of the interface robust or to modify a particularly troublesome aspect of the design. We got to a few moderate importance problems, and even a couple of low importance ones (e.g., the black cursor on black background). In all, we made changes associated with 39 of the problems on our list of 54, until the 60 hours of implementer time available for changes were allocated.

4.7. Redesigning and Implementing the Revised Interface

Much of the work for this final phase of formative evaluation was done when design solutions were proposed for each observed problem. At this point, we need only to update the appropriate design documentation to reflect our decisions, and resolve any conflicts or inconsistencies in the interaction design that might have resulted from our decisions. This is the time, of course, when we realize the full benefits of empirical formative evaluation, moving out of the current cycle of evaluation and into the subsequent cycle of (re)design and (re)evaluation.

5. CONCLUSIONS

What we have been talking about here, of course, is the very heart of the user interface development process. This involves the management of very difficult decision-making as to which problems are most important in terms of interface usability. But the point is that we are, in fact, engineering the interface, striving for achievement of our usability goals, rather than perfection. This approach recognizes diminishing economic returns in attempting to achieve perfection by trying to solve all known usability problems in a user interaction design. Therefore, usability management includes quantitative techniques for making decisions about which changes to make to a design as a result of a cycle of formative evaluation.

Compared to the top-down waterfall process often used in software engineering, where management can sign off on each phase, the iterative process of interface development and formative evaluation we have described is potentially a cycle that never ends. A development process that does not have a well-defined ending point is unacceptable to most managers. A control mechanism is required, one that will help managers (and developers) know what design changes are most cost effective in meeting usability goals, whether the iterative process is converging toward a usable interface, and when to stop iterating. Without such a control mechanism, interaction developers and evaluators can be caught in an infinite loop, thrashing about without any guidance and actually producing a worse interface (in terms of its usability) through this cyclic procedure. The control mechanism is the combination of a set of techniques including usability specifications, empirical formative evaluation, and cost/importance and impact analyses. Usability specifications set quantitative target expectations for user performance and satisfaction, and formative evaluation with representative users provides data to compare actual usability of the interaction design with these specifications. Cost/importance analysis and impact analysis are used to determine which interface problems observed during evaluation to address to get the greatest improvement in usability. Finally, of course, when usability specifications are met, the development team can be confident that the desired usability for the system has been achieved, and iteration can cease.

Some developers might hesitate to add these empirical formative techniques to an already over-burdened system and interface development process with limited financial and temporal resources. However, we know of no other approach that provides empirically-based, quantitative data for managing the iterative refinement process, effectively evaluating an interface, and thereby ensuring usability in the final product.

6. FUTURE TRENDS IN USER INTERFACE EVALUATION

We have, thus far in this paper, described state-of-the-art techniques used in formatively evaluating a user interface. Numerous avenues of exploration are open to the human-computer interaction world, to establish new techniques and trends in interface evaluation. We close by briefly discussing some of those we perceive as most important for continued advancement.

Strengthen analytic techniques with empirical observation.

In Section 3.3 we critiqued analytic evaluation techniques. We do not conclude, however, that the basic concept behind analytic techniques (inspecting and manipulating a representation of the interface design to identify usability flaws) should be discarded. Instead, the drawbacks of existing analytical techniques can be addressed in new approaches. For example, many of the purely analytical approaches to modeling for interface design evaluation suffer from lack of accurate task descriptions. Empirical observation of users is especially effective at capturing descriptions of the methods by which users actually perform tasks. A trend toward inclusion of empirical approaches in analytic methods will help overcome some of the weaknesses of these analytic methods.

Adapt global analysis techniques to situational analysis.

Complete, global modeling of an interaction design is costly, and requires persons skilled in the particular modeling technique being used. Finding variations of analysis techniques that support a focus on situations involving known usability problems can make analysis more cost effective. With such approaches, not all tasks must be described. Rather, situational analysis is used to address only tasks that are observed during formative evaluation to be problematic, those a user had difficulty with or failed to accomplish. Then, for example, tasks for which user performance met established usability goals do not have to be described (modelled). This trend will result in a savings of development time and cost.

Extend evaluation techniques to identify specific usability problems and suggest appropriate solutions.

In Section 2.4, we briefly discussed Scriven's concepts of intrinsic and payoff evaluation. Carroll, Singley, and Rosson (1992) describe the drawback Scriven observed for pay-off evaluation: while this kind of evaluation can reveal that something is wrong (in the current context, with the user interaction design), it cannot suggest what might be the cause or how it might be corrected. This leaves no structured way to deal with the result of evaluation. However, this drawback applies mainly to quantitative data. In Section 4, we showed that qualitative data collected during empirical formative evaluation can, indeed, lead to attribution of a usability problem—at least to its general cause, if not specific location—in a design. Nonetheless, the iterative refinement cycle has a missing link in that there are no techniques to suggest solutions to usability problems that are identified through formative evaluation. At this point designers are usually asked to synthesize a design change to fix the problem. The development process needs a way to close the iteration loop by providing designers with a principle- or theory-based approach to attacking redesign. There is a need for techniques that can assign credit and blame, pinpointing why user performance does not meet usability goals in terms of specific interface design flaws and shortcomings.

In answer to this need, we are seeing the beginnings of a trend that makes use of a combined approach to evaluation. To call on the strengths of both intrinsic and pay-off approaches, Scriven proposes a combined approach that he calls mediated evaluation (Carroll, et al., 1992, p. 5). The task-artifact framework of Carroll and Rosson, discussed in Section 3.2, is an example of mediated evaluation in the user interface context.

Develop more efficient, high impact usability evaluation techniques, to increase the likelihood of their use within real development projects.

We do not yet know enough about what makes good formative evaluation work and bad formative evaluation not work. At this point in time, formative evaluation is still more an art than a science. Many developers who are hesitant to perform formative evaluations often claim that it is largely because the process is too time-consuming. Thus, there is a pressing need for new, more precise, techniques that assist in assessing user performance and usability as quickly and effectively as possible. We need ways to formalize and codify evaluation processes so they can be brought into an evaluation methodology applicable across a broad spectrum of situations. Methods are also needed to manage the problem of prioritizing, for optimum allocation of finite future redesign effort, design problems discovered in evaluation. We must develop techniques for more efficient extraction of the most relevant usability data during empirical formative evaluation. Nielsen's discount usability engineering (1989) and cognitive "jogthroughs" (Rowley & Rhoades, 1992) are examples of some developments in improving the efficiency and impact of formative evaluation. Such trends will make the formative evaluation process more effective, thereby increasing the use in real world development environments.

Expand the formative evaluation venue, from the usability lab into a real world setting.

Empirical formative evaluation must escape the confines of the usability laboratory. Thomas and Kellogg (1989) warn that laboratory testing, while a very 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. We see a trend toward developing new techniques for capturing and analyzing data captured in situ, in order to evaluate a design in the most realistic setting.

Develop better methods for setting the most effective usability specifications.

Less ad hoc, more scientific approaches to establishing quantitative usability goals are needed. Currently, a drawback of usability specifications is the subjectivity of setting levels (worst, planned, best) for each attribute. In order to improve the process of creating and applying usability specifications in the development process, we need a better understanding of the true nature of usability itself. This trend could lead to improved ways of formulating usability specifications that more directly address the usability problem in the practical arena of product development. Perhaps as the field matures, we will accumulate a knowledge base of situations and applications, and feasible usability specifications that apply, along with rationale and guidelines for setting these specifications.

Most usability specifications are centered on measurable user task performance such as timing and error rates, reflecting that user performance is a big economic factor. But in the consumer marketplace it is hard to escape the conclusion, based on how vendors allocate their resources, that user preferences and opinions are considered more 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. This appears to be an area that could benefit from user acceptance prediction methods such as those we mentioned in Section 2.3.

Improve tools for rapid prototyping of designs.

To ensure usability of an interface, data obtained from evaluation with representative users is needed before much time and effort are invested in design and hopefully almost none invested in implementation. In order to have something to evaluate early in the development process, a prototype often must be used in place of the real system. Although we have not mentioned rapid prototyping very much in this paper, it is a well-recognized key activity in the iterative star life cycle and often a necessity for early empirical formative evaluation of the product. Rapid prototyping allows users to take a new design "for a spin," allows early observation of user behavior and performance, and encourages user participation and involvement, providing a concrete baseline for communication between developers and users.

However, developers ready to jump on the prototyping bandwagon should be aware of its limitations. For example, using a prototype leads to an evaluation of only the prototype. The higher the fidelity of a prototype to the appearance and behavior of the real system, the more effective its contribution to the real evaluation effort that it supports. Also, unfortunately, current prototyping tools are often limited in the range of applications for which they can be used to produce a prototype. Tools are typically built around a specific look and feel (e.g., Motif, Macintosh), making it difficult to deviate from those established interaction styles. And there is the very real problem of non-availability of good rapid prototyping tools on large workstations, a platform on which much interactive system development is occurring.

Many of the pitfalls associated with prototyping, especially rapid prototyping, originate from misconceptions of, misunderstandings of, and lack of agreement about the role of a prototype within a development project. Lack of understanding and failure to obtain agreement about the role of a prototype can lead to overpromising (including a feature in the prototype that will not be in the real system), loss of discipline ("it's only a toy system, who needs a methodology?"), or overworking the prototype (falling in love with the prototype and polishing it after it has served its usefulness for usability evaluation). In addition, management can view prototyping activity as wasteful or, conversely, can view the prototype as the final product and want to rush it to market.

Because of such shortcomings, there is a significant need for advancement of state-of-the-art prototyping approaches and tools. An important trend is to eliminate 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 prototype and real system—higher fidelity in the prototype and therefore more realistic results from formative evaluation.

Develop and evaluate tools for supporting the formative evaluation process.

We strongly believe it is unlikely that we will ever come close to automating all evaluation of interface designs; purely analytic approaches that replace empirical observations of real users performing real tasks do not seem feasible. There are, however, some analytic approaches that provide unusual approaches to analyzing user session data. One such technique is called maximal repeating patterns, or MRPs (Siochi & Ehrich, 1991), in which repeating user action patterns of maximum length are extracted from a user session transcript, based on the hypothesis that repeated patterns of usage (e.g., sequences of repeated commands) contain interesting information about an interface's usability. Like other methods of instrumenting interfaces to collect user keystrokes, mouse clicks, and so on, the MRP technique produces voluminous data; only a prototype tool for automated extraction and evaluation of MRPs exists. Still, while the MRP technique does help pinpoint specific problems, it does not indicate how an interaction design should be modified to fix those problems.

Currently there are few tools available to support the formative empirical evaluation process we described in Section 4. Such tools are needed to assist evaluators in collection and analysis of observed qualitative and quantitative user data during evaluation sessions. The Interface Design Environment and Analysis Lattice, or IDEAL (Ashlund & Hix, 1992), is an interactive tool to support the process of interaction design and formative evaluation. Components of IDEAL are based on user task descriptions that comprise the design. IDEAL allows a developer to attach specific benchmark task instances to general user task descriptions. The developer can also create usability specifications, which are then stored and displayed in conjunction with the appropriate benchmark and other tasks. During formative evaluation, a developer/evaluator can record and associate observed quantitative (e.g., benchmark performance) and qualitative (e.g., critical incidents) data directly

Chin, J. P., Diehl, V. A., & Norman, K. L. (1988). Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 213-218.

Davis, F. D. (1989). Perceived Usefulness, Perceived Ease of Use. *MIS Quarterly, 13*(3), 319-340.

Dick, W., & Carey, L. (1978). *The Systematic Design of Instruction.* Glenview: Scott, Foresman.

Gilb, T. (1981). *Design by Objectives.* (Unpublished).

Gong, R., & Elkerton, J. (1990). Designing Minimal Documentation Using a GOMS Model: A Usability Evaluation of an Engineering Approach. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 99-106.

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.

Gray, W. D., John, B. E., & Atwood, M. (1992). The Precis of Project Ernestine or an Overview of a Validation of GOMS. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 307-312.

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,* Amsterdam: North-Holland: Elsevier Science Publishers,

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.

Hix, D., & Hartson, H. R. (1993). *Developing User Interfaces: Ensuring Usability Through Product and Process.* New York: John Wiley & Sons, Inc.

Jeffries, R., & Desurvire, H. (1992). Usability Testing vs. Heuristic Evaluation: Was There a Contest? *ACM SIGCHI Bulletin, 24*(4), 39-41.

Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. M. (1991). User Interface Evaluation in the Real World: A Comparison of Four Techniques. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 119-124.

John, B. E. (1990). Extensions of GOMS Analyses to Expert Performance Requiring Perception of Dynamic Visual and Auditory Information. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 107-115.

John, B. E., & Vera, A. H. (1992). A GOMS Analysis of a Graphic, Machine-Paced, Highly Interactive Task. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 251-258.

Karat, C.-M., Campbell, R., & Fiegel, T. (1992). Comparison of Empirical Testing and Walkthrough Methods in User Interface Evaluation. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 397-404.

Kieras, D., & Polson, P. G. (1983). A Generalized Transition Network Representation for Interactive Systems. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM,

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

Kieras, D. E. (1988). Towards a Practical GOMS Model Methodology for User Interface Design. In M. Helander (Ed.), *Handbook of Human-Computer Interaction* Elsevier Science Publishers B. V.

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, 235-242.

Mackay, W. E., & Davenport, G. (1989). Virtual Video Editing in Interactive Multimedia Applications. *Communications of the ACM*, *32*(7), 802-810.

Moran, T. P. (1980). A Framework for Studying Human-Computer Interaction. In e. a. Guedj (Ed.), *Methodology of Interaction* (pp. 293-301). North-Holland Publishing Co.

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.

Nielsen, J. (1987). Using Scenarios to Develop User Friendly Videotex Systems. *Proceedings of NordData87, Joint Scandanavian Computer Conference*,

Nielsen, J. (1988). Evaluating the Thinking Aloud Technique for Use by Computer Scientists. *Proceedings of IFIP W. G. 8.1 International Workshop on Human Factors of Information Systems Analysis and Design*,

Nielsen, J. (1989). Usability Engineering at a Discount. In G. Salvendy, & M. J. Smith (Ed.), *Designing and Using Human-Computer Interfaces and Knowledge-Based Systems* (pp. 394-401). Amsterdam: Elsevier Science Publishers.

Nielsen, J. (1992). Finding Usability Problems Through Heuristic Evaluation. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 373-380.

Nielsen, J., & Molich, R. (1990). Heuristic Evaluation of User Interfaces. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 249-256.

Olson, J. R., & Olson, G. M. (1990). The Growth of Cognitive Modeling in Human-Computer Interaction Since GOMS. *Human-Computer Interaction*, *5*, 221-265.

Payne, S. J., & Green, T. R. G. (1986). Task-Action Grammars: A Model of the Mental Representation of Task Languages. *Human-Computer Interaction*, *2*, 93-133.

Peck, V. A., & John, B. E. (1992). Browser-SOAR: A Computational Model of a Highly Interactive Task. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 165-172.

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

Reisner, P. (1983). *Analytic Tools for Human Factors of Software* (RJ 3808 (43605)). IBM Research Laboratory, San Jose, CA.

Rowley, D. E., & Rhoades, D. G. (1992). The Cognitive Jogthrough: A Fast-Paced User Interface Evaluation Procedure. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 389-395.

Scriven, M. (1967). The Methodology of Evaluation. In R. Tyler, R. Gagne, & M. Scriven (Ed.), *Perspectives of Curriculum Evaluation* (pp. 39-83). Chicago: Rand McNally.

Siochi, A. C., & Ehrich, R. W. (1991). Computer Analysis of User Interfaces Based on Repetition in Transcripts of User Sessions. *Transactions on Information Systems*, ,

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

Wharton, C., Bradford, J., Jeffries, R., & Franzke, M. (1992). Applying Cognitive Walkthroughs to More Complex User Interfaces: Experiences, Issues, and Recommendations. *Proceedings of CHI Conference on Human Factors in Computing Systems*, New York: ACM, 381-388.

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

Williges, R. C. (1984). Evaluating Human-Computer Software Interfaces. *Proceedings of International Conference on Occupational Ergonomics*,

Wilson, M. D., Barnard, P. J., Green, T. R. G., & Maclean, A. (1988). Knowledge-Based Task Analysis for Human-Computer Systems. In G. C. v. d. Veer, T. R. G. Green, J.-M. Hoc, & D. M. Murray (Ed.), *Working with Computers: Theory Versus Outcome* (pp. 47-87). London: Academic Press.

jo-anne, per our conversation today, i am reporting that this tr is now published as: Hix, D. & Hartson, H. R. (1993). Formative Evaluation: Ensuring Usability in User Interfaces. Chapter 1 In L. Bass & P. Dewan (Ed.), Trends in Software, Volume 1: User Interface Software (pp. 1-30). New York: Wiley.

> TR-92-60 - Hix and Hartson, Formative Evaluation: Ensuring Usability in User
>   Interface (Development?)
-rex