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.
INTRODUCTION

DEBORAH HIX & REX HARSON

ENSURING USABILITY IN USER INTERFACES

FORMATIVE EVALUATION:
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, 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
Simply, each development activity should be followed by evaluation or some sort. Single starting point, nor is there a prescribed order for development activities, requirements/specification, not just in design. It also shows that there is not a prerequisite development activity (e.g., task/functionality/user analysis).

In contrast, evaluation (of a kind appropriate to each activity) is to be applied to every user interface/cycle (or a kind appropriate to each activity) is to be applied to every user interface/cycle (or a kind appropriate to each activity) is to be applied to every user interface/cycle (or a kind appropriate to each activity) is to be applied to every user interface/cycle (or a kind appropriate to each activity) is to be applied to every user interface/cycle (or a kind appropriate to each activity) is to be applied to every user interface/cycle (or a kind appropriate to each activity).

Figure 1. Star Life Cycle (adapted from Harrison & Hix, 1989)

(Harrison, et al., 1989) shown in Figure 1 illustrates a life cycle concept we call the star life cycle. In response to the need for a focus on interaction and correspondence on features, in response to the need for a focus on interaction and correspondence on features.
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.
and those methods are the topic of this paper.

Information design is one of the key elements that differentiates software systems from hardware ones. In this context, the user interface is a crucial component of the system.

Thus, software evaluation techniques must be appropriate for evaluating usability.

For our purposes here, since it is the interaction design that is evaluated for usability, the user interface could be considered in terms of artifacts, but it does not have to be. The interface does not necessarily produce output interface software in the usual sense. In the case of a WYSIWYG rapid prototyping environment, usability is implemented, executed, observed, and evaluated. But an interface need not necessarily be implemented, executed, observed, and evaluated. In user interfaces, user interface software is the medium through which an application's output is presented and devices that are presented to the user and with which the user interacts. The user interface component includes the icons, text, sounds, and graphics that are visible and heard and does while interacting with the system. In this context, the user interface is evaluated for usability. Rather, the interaction design is evaluated for usability.

So, surprising as it might be to those who have not thought about the distinction, it is.

This is true of goals.

Goals of usability, obviously, also software design. These same goals apply. Usability is not on freedom from bugs, functionality, and so on. Since the user interface software has many goals, including fidelity of implementation to the design, reliability, and the computational (non-interface) software component, software evaluation can be computationally intensive from software engineering and appropriate for developing well-known techniques for software engineering.
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 Kobart (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
provides a foundation based on psychological research for purposes of prediction.

The GOALS (Goals, Operators, Methods, and Selection Rules) model (Gard, et al., 1983) supports predictive evaluation through a systematic analysis of an interaction design.

measures of human performance.

of the prediction, each analytic method must be validated with empirical
design. Followed up later with empirical evaluation. By necessity, for the credibility
of the analytic evaluation is used, it is often used early in the evolution of an
complementary to, and not a complete substitute for, empirical evaluation (Reiser). Some
point in the development of an analytic task, however, analytic methods must be considered
independently. Techniques are sometimes considered a substitute for empirical evaluation, at least at
stages of the design, with human participants there, an analytic approach can be used to support a
result is a prediction of what would happen if usability were measured empirically.

The analytic approach is evaluation of user interaction design is generally based on
analytic evaluation of a user interaction design.

 Grammars (TAG) (Payne & Green, 1986). These models all are the basis for some sort of
the Command Language Grammar (CLG) (Morena, 1989); the keystroke-level model
the Command Language Grammar (CLG) (Morena, 1989); the keystroke-level model
the Command Language Grammar (CLG) (Morena, 1989); the keystroke-level model
Before we concentrate on empirical approaches to formative evaluation, we offer a
brief overview of some representative approaches to analytic evaluation, including

3.1 Some Examples of Analytic Evaluation

3. ANALYTIC FORMATIVE EVALUATION

Techniques for reaching that goal, as we will discuss in the remainder of this paper,
enable usability in a user interaction design. However, they are very different.
Both analytic and empirical evaluation in the context of HCI have the same goal:
knowledge a little bit at a time.
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
analysed can be overwhelming.

particular keywords. Like COMS and the CTO, the level of detail in a keyword action. Another parameter includes mental preparation (e.g., deciding to make a movement), pointing (e.g., moving hands to the keyboard), and simple drawings (e.g., cursor.

commands are specified as a series of commands. Predictions of user performance are made by a method for performing a task is the same as in the system design tool to predict expert user command language system is intended as a system design tool to predict expert user

generality. A model of the user, derived from a structural representation of a system, includes other simple physical actions (e.g., mouse clicks) at the same level of user-system interaction as the name implies—the level of keywords and also

The key-stroke-level model (Carr, et al., 1980) as discussed in Reiser (1983), provides a technique for comparing interaction designs (Wilson, et al., 1988)

past decade, there seems to be some doubt about the utility in evaluation of real

much discussion and acceptance in human-computer interaction literature. While the CTO concept has received
describing many aspects of a user interface. Like COMS, however, creating a CTO

between the two (Wilson, Buxton, Green, & Maclean, 1988). p. 23) describes human
of a system from its specific command language and to separate the conceptual model
representation. Essentially, the CTO is a design tool to separate the layers of
components, and these components are levels are intended to mirror the layers of
components. The assumption is that users form a layered mental representation of

The Reiser Action Language Model (1983) provides a technique for comparing

and user as two cooperating and communicating information processors. Users are not likely to make mistakes. Like the CTO, the Action Language models computer
make comparisons of alternative designs and to identify design choices that could
in the Action Language and then applying metrics to predict user performance to
apply this model to the Robotic Graphics System Interface, describing its interface

more view of expressions in the language of the user. Complexity measures of the user are
designed decisions that might cause user errors. Physical actions of the user are
predicted user performance associated with alternative designs, and for identifying

The Reiser Action Language Model (1983) provides a technique for comparing

analysed can be overwhelming.

particular keywords. Like COMS and the CTO, the level of detail in a keyword action. Another parameter includes mental preparation (e.g., deciding to make a movement), pointing (e.g., moving hands to the keyboard), and simple drawings (e.g., cursor.

commands are specified as a series of commands. Predictions of user performance are made by a method for performing a task is the same as in the system design tool to predict expert user command language system is intended as a system design tool to predict expert user

generality. A model of the user, derived from a structural representation of a system, includes other simple physical actions (e.g., mouse clicks) at the same level of user-system interaction as the name implies—the level of keywords and also

The key-stroke-level model (Carr, et al., 1980) as discussed in Reiser (1983), provides a technique for comparing interaction designs (Wilson, et al., 1988)

past decade, there seems to be some doubt about the utility in evaluation of real

much discussion and acceptance in human-computer interaction literature. While the CTO concept has received
describing many aspects of a user interface. Like COMS, however, creating a CTO

between the two (Wilson, Buxton, Green, & Maclean, 1988). p. 23) describes human
of a system from its specific command language and to separate the conceptual model
representation. Essentially, the CTO is a design tool to separate the layers of
components, and these components are levels are intended to mirror the layers of
components. The assumption is that users form a layered mental representation of

analysed can be overwhelming.

particular keywords. Like COMS and the CTO, the level of detail in a keyword action. Another parameter includes mental preparation (e.g., deciding to make a movement), pointing (e.g., moving hands to the keyboard), and simple drawings (e.g., cursor.

commands are specified as a series of commands. Predictions of user performance are made by a method for performing a task is the same as in the system design tool to predict expert user command language system is intended as a system design tool to predict expert user

generality. A model of the user, derived from a structural representation of a system, includes other simple physical actions (e.g., mouse clicks) at the same level of user-system interaction as the name implies—the level of keywords and also

The key-stroke-level model (Carr, et al., 1980) as discussed in Reiser (1983), provides a technique for comparing interaction designs (Wilson, et al., 1988)

past decade, there seems to be some doubt about the utility in evaluation of real

much discussion and acceptance in human-computer interaction literature. While the CTO concept has received
describing many aspects of a user interface. Like COMS, however, creating a CTO

between the two (Wilson, Buxton, Green, & Maclean, 1988). p. 23) describes human
of a system from its specific command language and to separate the conceptual model
representation. Essentially, the CTO is a design tool to separate the layers of
components, and these components are levels are intended to mirror the layers of
components. The assumption is that users form a layered mental representation of

analysed can be overwhelming.

particular keywords. Like COMS and the CTO, the level of detail in a keyword action. Another parameter includes mental preparation (e.g., deciding to make a movement), pointing (e.g., moving hands to the keyboard), and simple drawings (e.g., cursor.

commands are specified as a series of commands. Predictions of user performance are made by a method for performing a task is the same as in the system design tool to predict expert user command language system is intended as a system design tool to predict expert user

generality. A model of the user, derived from a structural representation of a system, includes other simple physical actions (e.g., mouse clicks) at the same level of user-system interaction as the name implies—the level of keywords and also

The key-stroke-level model (Carr, et al., 1980) as discussed in Reiser (1983), provides a technique for comparing interaction designs (Wilson, et al., 1988)

past decade, there seems to be some doubt about the utility in evaluation of real

much discussion and acceptance in human-computer interaction literature. While the CTO concept has received
describing many aspects of a user interface. Like COMS, however, creating a CTO

between the two (Wilson, Buxton, Green, & Maclean, 1988). p. 23) describes human
of a system from its specific command language and to separate the conceptual model
representation. Essentially, the CTO is a design tool to separate the layers of
components, and these components are levels are intended to mirror the layers of
components. The assumption is that users form a layered mental representation of
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
By separating empirical studies (Nielsen, 1999) in which the users had no usability problems, independent evaluations effectively solved this problem. This approach was validated by subsequent evaluations effective at uncovering usability design problems.

Scientists and designers seek ways to make their products usable, effective, and efficient. One way to achieve this is to design products based on empirical evidence. However, empirical evidence alone may not be sufficient to ensure usability. Designers and evaluators need to follow a set of guidelines to ensure that the products they create are usable.

Nielsen (1993) provides a set of guidelines for design and evaluation that can help ensure usability. These guidelines include:

1. **Familiarity**: Ensure that the product is familiar to users.
2. **Visibility**: Ensure that the interface is visible to users.
3. **Consistency**: Ensure that the interface is consistent across the product.
4. **Feedback**: Ensure that the interface provides feedback to users.
5. **Help**: Ensure that the interface provides help to users.

These guidelines can be used in the design and evaluation process to ensure that products are usable.
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.
This supports determination of the effects on user performance of a given change in
plus response time of the system, are represented on a critical path chart.
(1990)). In this method, cognitive, perceptual, verbal, and motor actions for a task,
activities (C. Gray, John A. Wood, 1999; C. Gray, John S. Lerman, 1999; A. Wood,
performance, allowing analysis to sort out the complexity of parallel operations.
method of project management, was shown to be remarkably accurate in predicting
application. CPM-GOMS, which is GOMS analysis combined with the critical path
and repetitive tasks such as those of a telephone assistance operator. In such an
effective for real-world problems within specific classes of tasks (C. Gray, 1999).
However, in all fairness, GOMS-based analytic evaluation has been shown to be

Because of these limitations, we do not foresee analytic formative evaluation
presented with the software of many of these analytic evaluation techniques.
area of human-computer interaction research, we find the still of issues just
while analytic evaluation of user interactions has been, and continues to be, an open
Feedback Cycle.
Incremental user actions consist of perception and reaction in a closed-loop
model. We seek is a model of highly interactive direct manipulation involving
sequences wherein the user controls a command, then the system executes it. The
Many of these approaches are based on an open-loop model of command

Independent of the overall design—would be more cost effective.
approach supporting local analysis—enables the use of one area of design at any level can proceed. An
model built for the entire interface before analysis, at any level, can proceed. An
most analytic approaches require that the design be complete and a full global
necessary to provide a complete user task-oriented representation of a design.
representation that includes a task structure, and none deal with the temporal relations
While most of the analytic approaches mentioned above produce a
reveals the greatest indicators of usability problems.
interface system. And if it is, indeed, study of error-tolerant situations has often
but cognitive performance is obviously not the typical experience of a user of an
Almost all analytic approaches assume error-free, expert task performance.
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
Developing and Conducting the Experiment

4.3. Developing and Conducting the Experiment

Or simply the evaluator.

Is primarily responsible for these activities as the user interaction design evaluator, performing any one of these steps at various times, we will refer to the person who while may be involved in many members of an interaction development team may be involved in 1993.

Steps' including hands-on exercises and solutions, can be found in (Hix & Hartson).

Deeper coverage of each of these activities—the formative evaluation process—in-depth coverage of each of these activities—allows collection of data that can be analyzed to determine usability of the user interface the reader an appreciation for developing an experiment that overviews to give the reader an appreciation for developing an experiment that.

We do not intend, in this paper, to give details on each step; we will present an overview of.

Redesigning and implementing the revised interface

Drawing conclusions

Analyzing the data

Generalizing and collecting the data

Developing and conducting the experiment

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

Steps in Empirical Formative Evaluation

on Qualitative Data Collection Techniques, below.

Qualitative data, e.g., critical incidents and verbal protocols, discussed in Section 4.2.

Analytical techniques (e.g., benchmark tasks, measurements) can produce

Performance scales or questionnaires), can also produce qualitative data. Also,

Analytical and formative evaluation with qualitative data, analytical and formative evaluation (e.g., using user

Even though often associated with qualitative data and

Cycles of iterative development

Which design features are associated with measured usability problems during all
• 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
Without measurable specifications, it is impossible to develop usable interfaces. Appropriate benchmarks help to determine quantifiable usability, as stressed by Carroll and Rosson (1989) in their process. Carroll and Rosson (1989) also stressed the need for quantifiable usability.

1986, is a process through which usability characteristics are specified, and (Whitbread, et al, 1988), usability engineering, as defined in (Good, et al, 1986) are woven and integrated into usable software development. These ideas were refined and integrated into the usability engineering process by (Good, et al, 1981) and (Benne, 1964) adapted this approach to usable specification as a technique for Define the concept of formal usable specification in a particular form with various metrics. The concept of formal usable specification is an important part of usability and user performance.

4.3.2 Developing Usability Goals and Experimental Tasks

new and/or severe problems.

The participations and their additional participations are less and less likely to uncover the participations, and their additional participations are discovered with low or ineffective number. Empirical work by Yitzhak (1992) corroborates these observations, offering insights into participations. Other lists participations per well-defined user class is the most cost-effective number. Emphasis on participations per well-defined user class is not worth the time and effort involved. But more than ten participations per class is not worth the evaluation is niche to five per user class. Only one participations per user class is necessary for a well-defined user class, but more than ten participations per class is not worth the time and effort involved. But more than ten participations per class is not worth the time and effort involved. But more than ten participations per class is not worth the time and effort involved.
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.

<table>
<thead>
<tr>
<th>Usability Attribute</th>
<th>Measuring Instrument</th>
<th>Value to be Measured</th>
<th>Worst Acceptable Level</th>
<th>Planned Target Level</th>
<th>Best Possible Level</th>
<th>Current Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial use</td>
<td>&quot;Create a line drawing&quot; benchmark task</td>
<td>Length of time to create line drawing on first trial</td>
<td>30 secs.</td>
<td>15 secs.</td>
<td>5 secs.</td>
<td>10 secs.</td>
</tr>
<tr>
<td>First impression</td>
<td>Questionnaire</td>
<td>Average rating (range -2 to 2)</td>
<td>0</td>
<td>0.5</td>
<td>1.25</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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
example, for the usability goal desired for "first impression" of the graphics
second, while the very best time to perform it is estimated at 5 seconds. A similar
The expected (planned) length of time to perform this task is 15
the task to "create a line drawing" must be at the longest (worst level), 30 seconds.
As shown in Figure 3, for this simple example, we state that initial use time to perform

shape of this sketch:

description might be: "Produce a line drawing similar to the approximate size and
evaluation session. For example, for the "create a line drawing" benchmark, the task
benchmark task must be written telling the user what to do (but not how) during an
during initial use of the interface for the specific benchmark task to "create a line
the first (initial) time they use it. Thus, in the first attribute, user performance
can perform tasks usability attribute for virtually any interface is how quickly users can perform tasks
Let us briefly explain the usability specifications shown in Figure 3. An important

Application

Figure 3. Examples of usability specifications for a graphical drawing

attribute (when available)

current level is the currently known level of the value to be measured for the
level

best possible level is a realistic slice-of-the-car upper limit of the inspiration
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.
In the evaluation, the system must be evaluated to find.

Evaluated the system, not to evaluate them.

It is important to make very clear to the participants that the purpose of the session is to
will be expected to do, and the procedures to be followed by the user. It is also
will do about the interface the participant will be using, since what the user

prepare introductory instructional remarks that will be given uniformly to each

In conjunction with developing experimental procedures, the evaluator should

be sufficient.

In this case, laboratory testing may have

the two is the ideal circumstance for formal evaluation, but in real life, the field
design are typically discovered. Field testing works well for larger cycles, when
collect cycles of formal evaluation, when major problems with an interaction
the situation is more realistic. Laboratory testing is typically more appropriate for
the evaluator can have greater control over the experiment, but the conditions are

field testing each have pros and cons. In a laboratory setting, the

more qualitative, longer-term data are often collected.

normal working environment in which the user is expected to use the interface, and
picking the interface to be evaluated, the current version is set up in situ, in the
insertion involves

lab setting where they perform the benchmark tasks, performance measures are
involves bringing the user to the interface, that is, users are brought into a usability

laboratory testing or field testing, or both, will be performed. Laboratory testing

during a test session with a participant. The evaluator must decide on whether

in order to properly direct an experimental session, the evaluator must produce

4.3.3 Directing an Experimental Session
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:
performance by a participant. The evaluator will find that some participants are not
during the use of the system or during other non-imposed procedures.
might be done to fix these problems. The verbal protocol technique is best employed
immensely effective in determining what problems a participant is having and what
should be used sparingly with benchmark tasks where timing is important. But it is
baptised, and so on. This technique, obviously, is invasive to the participant, so it
having a problem, when they expected to happen, but didn't, when they wished that
talk out loud while working, inquiring when they are living to do, or why they are
protocol driving, also called "thinking aloud." Here, the evaluator asks a participant to
Perhaps the most common technique for qualitative data generation is verbal

- Structured Interviews
- Critical Incident taking and
- Verbal Protocol taking

dare include the following:

desire, the kinds of techniques that are most effective for generating qualitative
data are extremely important for formative evaluation of a user interaction.

4.2. Qualitative Data Generation Techniques

Diethi (e Norman, 1988) is the best of these validated questionnaires.

The Quesionnaire for User Interface Satiastion, or QUIS. survey (China,
effective technique for producing qualitative data on subjective user opinion of an
administer, but is not easy to produce so that it is valid and reliable. It is also the most
interfere being evaluated. This kind of questionnaire or survey is not expensive to
process, scales, for different features that are relevant to usability of the
The second qualitative data generation technique is questionnaires, or user

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

- Benchmark tasks, and
- Questionnaires.
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.
The participant's face may be visible, including an expression. The camera aimed at the participants' hands and the screen is the most obvious. "Videocapture" was difficult, capturing detailed capture of what occurs during the video.

Videocapture is a well-known and frequently-used data collection technique.

Performing each task or using the interface freely.

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

In order to help take notes:

First few items, it is a good idea to have a second evaluator observe the session in a session. When an evaluator is directing a test session for the activities process during the session, because a minute of simultaneous activities to take copious notes, other with pencil and paper or using a word processor, as data capture during a usability evaluation session. The evaluator should be prepared for evaluators, their notetaking in real time is still the most effective technique to use for evaluators.

We have found, through our own experience and numerous conversations with other.

Gather qualitative data from participants during a formal evaluation experiment.

There are several recommendation techniques for capturing both qualitative and quantitative data. A data collection technique.

What did you like least? "How would you change so-and-so?" and so on.

Examples of such general questions as "What did you like best about the interface?"

Structured Interviews provide another form of qualitative data. Those are typically
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 automatically) user session transcripts. We will briefly
of the development team are involved.

which, at various times, all members of the development team are involved.

scheduled design and implementation solutions is essentially one of negotiation in

interface. The process of determining how to convey the collected data into

address solving those problems in order of their potential impact on usability of the

identification of the observed problems and potential solutions to them. We will then

further data analysis—which of which will be preliminary data analysis—is structured

cycle of testing. Then more in-depth data analysis should be performed. Our goal in

If usability specifications have not been met (the most likely situation after the first

should be more (or less) stringent.

the development team can should reassess the usability specifications to see if they

locution, and are not developing a good interaction of high usability. Then, obviously,

whatever reason, there is suspicion that the usability specifications may be too

satisfactory and information for this version can stop. The one exception is if for

usability of the current version of the design is acceptable, then the design is

enough planned target levels have been met to satisfy the development team that

this cycle of formative evaluation. If all worst acceptable levels have been met during

which usability specifications have been met; which have not been met during

observed results with the specified usability goals, the evaluator can immediately

results column added to the usability specification table. By directly comparing

and so on. The calculator can then enter a summary of the results into a spreadsheet

stands in the usability specifications (time, error counts, questionnaires, ratings,

cycles of formative evaluation.

not, how to modify the design to help us converge toward those goals in subsequent

(ANOVA) or t-tests or F-tests. Rather, we will be using some data analysis techniques

general, be performing inferential statistical analyses, such as analysis of variance

completed, the data collected during those sessions must be analyzed. We will not, in

After all evaluation sessions for a particular cycle of formative evaluation are

4.5 Analyzing the Data

Future Trends, below.

discuss tools, including those for analysis of user session transcriptions, in Section 6 on
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.

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

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**
appropriate, and different solutions may have very different costs associated with them. More than one possible solution is often possible, and a study of other similar designs, where data is available, may suggest solutions by analogy or through a variety of other means, including design techniques.

The development team must also propose one or more possible solutions, that is, their tasks.

mounting initial inclusion of cause and effect is difficult to incorporate, because of the consistent relationship below, the design effect of an efficient implementation of the established usability specification, for high importance, intuitive and effective upon pragmatic development concerns. First candidates include development team (possibly including some users), and can take into account many factors, including impact on overall system integrity and consistency. The importance ranking results from a holistic decision-making process among the design.

Changes, especially if the process is leading to convergence toward a more usable design, especially in the longer lists of problems. Lower cycles generally produce fewer problems, and the earlier cycles of formative evaluation especially give the most clear and immediate feedback. Two of these problems are shown in Figure 4. As we have already mentioned, efficient and effective application is important, after about four hours of the first cycle of evaluation sessions, we had a list of 45 problems to tackle, ranging from serious to superficial drawing application interface. After all four columns for each problem, the concept of problems—potentially hundreds of them—and as with usability specifications, a team of all is the cursor.

From whatever task they were using, do when a "wandering window" appeared, observed the user intensely disliked this design feature, because it distracted them purposefully place the window. The evaluator during calculation of several tasks,

the section following mouse movement, until the user clicked a mouse button to appear, it remained connected to the cursor, and the illogic wandered around on the screen. In this application, whenever a new window appeared, the first interface problem shown in Figure 4 is
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
As already mentioned, another kind of analysis, called impact and risk analysis (C. Good, et al.), can be used to determine what problems most affect achievement of the goal to remain unchanged, at least for the next cycle of evaluation. To make sure which problems were going to get addressed, and which were difficult, we had some difficulties. We had multiple suggestions. We had some suggestions for changes, even after selecting the lowest-cost suggestion to change for improvement. We decided the Cost column to find we had more than 140 hours (across two programmers) allocated for modifications in our first cycle of modifications. In our evaluation of the graphical drawing application, we had 60 resources (in particular: time and people) we have available to allocate for making changes as listed in a table like the one in Figure 4. We must first determine the importance of the problem, then a decision on usability and relative cost and importance. We can perform a couple of different kinds of analyses that help determine changes will have the greatest impact. 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 which changes will have the greatest impact.

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 which changes will have the greatest impact.
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.

<table>
<thead>
<tr>
<th></th>
<th>Quantitative (numeric)</th>
<th>Qualitative (non-numeric)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(measures performance)</td>
<td>- Impact analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- User performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>metric (benchmarking)</td>
<td></td>
</tr>
<tr>
<td><strong>Subjective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(involves opinion)</td>
<td>- Cost/importance</td>
<td>- Critical incident</td>
</tr>
<tr>
<td></td>
<td>analysis</td>
<td>analysis</td>
</tr>
<tr>
<td></td>
<td>- User satisfaction</td>
<td>- Protocol taking/analysis</td>
</tr>
<tr>
<td></td>
<td>metric (user preference)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Structured interviews</td>
</tr>
</tbody>
</table>

**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.
High cost problems are so critical that they must be fixed despite their importance. High cost problems at the top of the list get first priority, unless, for example, some of the high importance problems are higher in cost and can be addressed given the time and other resources allotted for modifications. We can start with high importance/high cost problems at the top of the list followed by high importance/moderate cost problems (descending). Low cost problems can be fixed as required.

We can finally now complete the Resolution Column of Figure 4. If the list is ordered in descending order of importance, we can determine how many problems were high, moderate, and low in cost. For example, if we are asked to fix problems with high cost, we can determine the number of high cost problems that need to be fixed.

4.6. Drawing Conclusions

In formal evaluation, we must understand the potential influence of qualitative data on our decisions. Qualitative data can provide insights into the potential impact of decisions, but it cannot replace quantitative data. We must evaluate both types of data to make informed decisions.
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
6. FUTURE TRENDS IN USER INTERFACE EVALUATION

That product, the final product, effectively evaluating an interface, and the ability to consume usability in the process, effectively evaluating an interface, and the ability to consume usability in the process. The user satisfaction with the system, as well as the development team, can be considered, that the desired satisfaction for the system has been achieved, and the decision can cease.

Get the greatest improvement in usability. Finally, of course, when usability specifications are used to determine which interface is most beneficial for decision-making. Usability, in general, usability, satisfaction, and performance evaluation with specifications for user performance and satisfaction, and formal usability evaluation with conformance to usability specifications, set qualitative targets.

In Section 3, we critiqued analytic evaluation techniques. We do not conclude,
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.
Testing does not lead to the kind of "colloquial validity" that comes from rich, empirical evaluation. Instead, addressing usability concerns is not enough. In particular, laboratory setups and Krolles (89) make that laboratory testing is a very productive step toward addressing usability concerns, is not enough. In particular, laboratory setups and Krolles (89) make that laboratory testing is a very productive step.

Empirical, formalistic evaluation must escape the confines of the usability laboratory. The real world setting.

Expand the formalistic evaluation concept from the usability lab into a

Increasing the use in real-world development environments.

These trends will make the formalistic evaluation process more effective, thereby leading us to improve the efficiency and impact of formalistic evaluation. Such developments bring us closer to the "truth" and reduce the bias of formalistic evaluation. Neilson (cited in Krolles, 1992) is an example of some successful empirical formalistic evaluation. Developers can be guided by using such development techniques for more efficient and effective development of software. The problems discovered in evaluation must be fixed, not ignored. Effective design is a result of the redesign effort, not a problem discovered in evaluation. We must also need to manage the problem of prioritization, for the allocation of resources is necessary to enable developers to develop methods to formalize methodologies applicable across a broad spectrum of situations. Methods need to be formalized and codified evaluation processes to ensure they can be brought into the research and development community. We need ways to formalize and codify evaluation processes so they can be brought into the research and development community.

Thus, there is a pressing need for new, more precise, techniques that assist in evaluation. What is true for evaluation is also true for effectiveness evaluations. More than a science, many development processes need to be more precise. At this point in time, formalistic evaluation is still the most effective way to measure the progress of a software development project. We do not yet know enough about what makes good formalistic evaluation work and how to increase the likelihood of their use within real development projects.

Interface context.

Rosson discussed in section 3.2 is an example of mediated evaluation in the user evaluation (Carrillo et al., 1992, p. 5). The last-mentioned framework of Carrillo and others proposes a combined approach that he calls mediated evaluation. To call on the strengths of both intuitive and pay-a-fee approaches, section proposes a combined approach that he calls mediated evaluation.
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
The goal of prioritization is to identify key priorities for development of the real system. The prioritization approach must follow the prioritization guidelines and tools. An important trend is to eliminate the need for activities to prioritize a project, especially rapid prioritization. However, developers need to jump on the prioritization bandwagon and put it into action. Providing a concise, clear, and consistent explanation of the project, user participation, and involvement, and an easy-to-use, efficient, and comprehensive prioritization tool is key to the prioritization process. It is important to note that the prioritization process is not limited to the real system, and should also be applied to the prioritization of the real system.)

This paper is a well-documented and comprehensive overview of the real system. Although we have not mentioned rapid prioritization very much in the development process, a prioritization process must be used in place to evaluate the project. In order to have something
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.

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.


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

-rex