Chapter 2

Usability Evaluation of User Interfaces

2.1 Introduction

Usability is the extent to which users can use a computer system to achieve specified goals effectively and efficiently while promoting feelings of satisfaction in a given context of use.[1] Usability evaluation (UE) consists of methodologies for measuring the usability aspects of a system's user interface (UI) and identifying specific problems [Dix *et al.* 1998; Nielsen 1993]. Usability evaluation is an important part of the overall user interface design process, which ideally consists of iterative cycles of designing, prototyping, and evaluating [Dix *et al.* 1998; Nielsen 1993]. Usability evaluation is itself a process that entails many activities depending on the method employed. Common activities include:

- **Capture -** collecting usability data, such as task completion time, errors, guideline violations, and subjective ratings;
- **Analysis -** interpreting usability data to identify problems in the interface; and
- **Critique -** suggesting solutions or improvements to mitigate problems.

A wide range of usability evaluation techniques have been proposed, and a subset of these are currently in common use. Some evaluation techniques, such as formal usability testing, can only be applied after the interface design or prototype has been implemented. Others, such as heuristic evaluation, can be applied in the early stages of design. Each technique has its own requirements, and generally different techniques uncover different usability problems.

Usability findings can vary widely when different evaluators study the same user interface, even if they use the same evaluation technique [Bailey *et al.* 1992; Desurvire 1994; Jeffries *et al.* 1991; Molich *et al.* 1998; Molich *et al.* 1999; Nielsen 1993]. As an example, two comparative usability testing studies (CUE-1 [Molich *et al.* 1998] and CUE-2 [Molich *et al.* 1999]) demonstrated less than a 1% overlap in findings among four and nine independent usability testing teams for evaluations of two user interfaces[2]. This result implies a lack of systematicity or predictability in the findings

[1] Adapted from ISO9241 (*Ergonomic requirements for office work with visual display terminals* [International Standards Organization 1999]).

[2] The first study involved four professional teams, while the second study involved seven professional teams and two student teams. Details were not provided about study participants.

of usability evaluations. Furthermore, usability evaluation typically only covers a subset of the possible actions users might take. For these reasons, usability experts often recommend using several different evaluation techniques [Dix *et al.* 1998; Nielsen 1993].

How can systematicity of results and fuller coverage in usability assessment be achieved? One solution is to increase the number of usability teams evaluating the system, and to increase the number of study participants. An alternative is to *automate* some aspects of usability evaluation, such as the capture, analysis, or critique activities.

Automating some aspects of usability evaluation has several potential advantages over non-automated evaluation, such as:

- Increasing consistency of the errors uncovered. In some cases it is possible to develop models of task completion within an interface, and software tools can consistently detect deviations from these models. It is also possible to detect usage patterns that suggest possible errors, such as immediate task cancellation.

- Increasing the coverage of evaluated features. Due to time, cost, and resource constraints, it is not always possible to assess every single aspect of an interface. Software tools that generate plausible usage traces make it possible to evaluate aspects of interfaces that may not otherwise be assessed.

- Enabling comparisons between alternative designs. Because of time, cost, and resource constraints, usability evaluations typically assess only one design or a small subset of features from multiple designs. Some automated analysis approaches, such as analytical modeling and simulation, enable designers to compare predicted performance for alternative designs.

- Predicting time and error costs across an entire design. As previously discussed, it is not always possible to assess every single aspect of an interface using non-automated evaluation. Software tools, such as analytical models, make it possible to widen the coverage of evaluated features.

- Reducing the need for evaluation expertise among individual evaluators. Automating some aspects of evaluation, such as the analysis or critique activities, could aid designers who do not have expertise in those aspects of evaluation.

- Reducing the cost of usability evaluation. Methods that automate capture, analysis, or critique activities can decrease the time spent on usability evaluation and consequently the cost. For example, software tools that automatically log events during usability testing eliminate the need for manual logging, which can typically take up a substantial portion of evaluation time.

- Incorporating evaluation within the design phase of UI development, as opposed to being applied after implementation. This is important because evaluation with most non-automated methods can typically be done only after the interface or prototype has been built and changes are more costly [Nielsen 1993]. Modeling and simulation tools make it possible to explore UI designs earlier.

It is important to note that automation is considered to be a useful *complement* and *addition* to standard evaluation techniques such as heuristic evaluation and usability testing – not a substitute. Different techniques uncover different kinds of problems, and subjective measures such as user satisfaction are unlikely to be predictable by automated methods.

1. Specify usability evaluation goals.
2. Determine UI aspects to evaluate.
3. Identify target users.
4. Select usability metrics.
5. Select evaluation method(s).
6. Select tasks.
7. Design experiments.
8. Capture usability data.
9. Analyze and interpret usability data.
10. Critique UI to suggest improvements.
11. Iterate the process if necessary.
12. Present results.

Figure 2.1: Activities that may occur during the usability evaluation process.

Despite the potential advantages, the space of usability evaluation automation is quite underexplored. This chapter presents a detailed survey of UE methods, with an emphasis on automation; a shorter version of this survey is scheduled for publication [Ivory and Hearst 2001]. The chapter begins with a brief overview of the usability evaluation process. It introduces a taxonomy for classifying UE automation and summarizes the application of this taxonomy to 133 usability evaluation methods. Several sections describe these methods in more detail, including summative assessments of automation techniques. The results of this survey suggest promising ways to expand existing approaches to better support usability evaluation; these approaches are also discussed.

2.2 The Usability Evaluation Process

Usability evaluation is a process that entails some of the activities depicted in Figure 2.1, depending on the method used. This section discusses each of these activities. Several literature sources informed this discussion, including [Dix *et al.* 1998; Nielsen 1993; Shneiderman 1998].

2.2.1 Specify Usability Evaluation Goals

Usability evaluation is applicable at all stages of a UI life cycle (e.g., design, implementation, and re-design). At these various stages, different UE goals are relevant. Below is a list of typical UE goals.

- Specify UI requirements
- Evaluate design alternatives
- Identify specific usability problems

- Improve UI performance

The evaluator must clearly specify the goals of the usability evaluation at the outset of the study. These goals influence other aspects of UI assessment, such as the UI components to evaluate and appropriate evaluation methods.

2.2.2 Determine UI Aspects to Evaluate

Some UIs can be extremely large and complex, and an evaluation of all aspects may not be economically feasible. Hence, the evaluator must determine specific UI aspects to evaluate. These aspects must be consistent with the goals of the usability evaluation.

2.2.3 Identify Target Users

An interface may be intended for a large user community, but it is important to determine user characteristics most relevant for the study and for the UI aspects in particular. If users are employed during the study, they need to be as representative of the larger user community as possible.

2.2.4 Select Usability Metrics

Usability metrics are a crucial component of the usability evaluation. The goal in selecting these metrics is to choose a minimal number of metrics that reveal the maximum amount of usability detail for the UI under study. ISO Standard 9241 [International Standards Organization 1999] recommends using effectiveness, efficiency, and satisfaction measures as described below.

- Effectiveness is the accuracy and completeness with which users achieve specified goals. Example metrics include: percentage of goals achieved, functions learned, and errors corrected successfully.
- Efficiency assesses the resources expended in relation to the accuracy and completeness with which users achieve goals. Example metrics include: the time to complete a task, learning time, and time spent correcting errors.
- Satisfaction reflects users' freedom from discomfort and positive attitudes about use of an interface. Example metrics include: ratings for satisfaction, ease of learning, and error handling.

Metrics discussed above are quantitative in nature. Non-quantitative metrics could include, for example, specific heuristic violations identified during a usability inspection.

2.2.5 Select Evaluation Method(s)

Choosing one or more usability evaluation methods is an important step of the UE process. There are five classes of UE methods: usability testing, inspection, inquiry, analytical modeling, and simulation. An inspection entails an evaluator using a set of criteria to identify potential usability problems in an interface, while testing involves an evaluator observing[3] participants interacting with

[3]During some usability testing, such as remote testing, the evaluator may not actually observe a participant interacting with an interface. Other techniques, such as logging (discussed in Section 2.5.2), may be employed to record the interaction for subsequent analysis.

an interface (i.e., completing tasks) to determine usability problems. Similar to usability testing, inquiry methods entail gathering subjective input (e.g., preferences) from participants, typically through interviews, surveys, questionnaires, or focus groups. Analytical modeling and simulation are engineering approaches to UE that enable evaluators to predict usability with user and interface models. Sections 2.5 – 2.9 discuss the five method classes as well as methods within each of the classes in more detail.

UE methods differ along many dimensions, such as resource requirements, costs, results, and applicability (i.e., at what stages of the interface development process). There is a wide range of methods that one could employ at all stages of system development, which actually complicates choosing an appropriate method. Human Factors Engineering, a company specializing in usability evaluation, has an online resource, Ask Usability Advisor [Human Factors Engineering 1999a], that recommends UE methods based on the following usability requirements: software development stage (requirement, design, code, test, and deployment), personnel availability (usability experts, participants, and software developers), usability dimensions to be measured (effectiveness, efficiency, and satisfaction), the need to obtain quantitative measures, and the need to do remote evaluation. There are also two comprehensive archives of UE methods online - Usability Evaluation Methods [Human Factors Engineering 1999b] and the Usability Methods Toolbox [Hom 1998].

UE methods uncover different types of usability problems; therefore, it is often recommended for evaluators to use multiple assessment methods [Jeffries *et al.* 1991; Molich *et al.* 1998; Molich *et al.* 1999; Nielsen 1993]. For example, during a usability test, participants may also complete questionnaires to provide subjective input; thus, enabling evaluators to gather quantitative and qualitative data.

2.2.6 Select Tasks

Tasks are the most crucial part of the usability evaluation [Dix *et al.* 1998; Nielsen 1993; Shneiderman 1998]. They must be appropriate for the UI aspects under study, the target users, and the evaluation method. Other constraints may affect the selection of tasks, such as cost and time limits during usability testing sessions, for instance.

2.2.7 Design Experiments

After completing the previously discussed activities, the evaluator may need to design experiments for collecting usability data. In particular, the evaluator needs to decide on the number of participants (evaluators and users), the evaluation procedure (this is largely dictated by the UE method) as well as on the environment and system setup. The nature of experiments depends on the evaluation method. Experiments may entail: completing tasks in a controlled manner (usability testing); responding to specific questions (inquiry); or comparing alternative designs (analytical modeling and simulation). It is also recommended that the evaluator conduct pilot runs during this phase [Nielsen 1993], especially if user involvement is required.

2.2.8 Capture Usability Data

During this phase, the evaluator employs the UE method to record previously specified usability metrics. For some methods, such as usability testing and inspection, the evaluator may also record specific usability problems encountered during evaluation.

2.2.9 Analyze and Interpret Data

The primary goal of usability data analysis is to summarize the results in a manner that informs interpretation. This summarization may entail statistical techniques based on the goals of the UE. It may also entail creating a list of specific usability problems found along with their severity.

Actually interpreting the results of the study is a key part of the evaluation. It entails using the analysis of usability data to draw conclusions as dictated by the evaluation goals. For example, it may mean concluding that one design is better than another or whether usability requirements have been met.

2.2.10 Critique UI to Suggest Improvements

Ideally, analysis and interpretation of usability data illustrate flaws in the UI design as well as ways to possibly improve the design. Subsequent analysis may be required to verify that suggested improvements actually improve interface usability.

2.2.11 Iterate Process

Analysis and interpretation of usability data may illustrate the need to repeat the UE process. This iteration may be warranted due to the identification of other UI aspects that need evaluation or changes to the UI. Hence, UE may consist of several cycles through this process. This is as expected when an evaluator follows usability engineering or iterative design processes [Dix *et al.* 1998; Nielsen 1993].

2.2.12 Present Results

The final step of the usability evaluation process is to communicate the results and interpretation of these results to the stakeholders. Ideally, the evaluator presents the results such that they can be easily understood (e.g., using graphs and providing severity ratings) and acted upon.

2.3 Taxonomy of Usability Evaluation Automation

In this discussion, a distinction is made between WIMP (Windows, Icons, Pointer, and Mouse) interfaces and Web interfaces, in part because the nature of these interfaces differ and in part because the usability evaluation methods discussed have often only been applied to one type or the other in the literature. WIMP interfaces tend to be more functionally-oriented than Web interfaces. In WIMP interfaces, users complete tasks, such as opening or saving a file, by following specific sequences of operations. Although there are some functional Web applications, most Web interfaces offer limited functionality (i.e., selecting links or completing forms), but the primary role of many Web sites is to provide information. Of course, the two types of interfaces share many characteristics; their differences are highlighted when relevant to usability evaluation.

Several surveys of UE methods for WIMP interfaces exist; Hom [1998] and Human Factors Engineering [1999b] provide a detailed discussion of inspection, inquiry, and testing methods (these terms are defined below). Several taxonomies of UE methods have also been proposed. The most commonly used taxonomy is one that distinguishes between predictive (e.g., GOMS analysis and cognitive walkthrough, also defined below) and experimental (e.g., usability testing) techniques [Coutaz 1995]. Whitefield *et al.* [1991] present another classification scheme based on the presence

or absence of a user and a computer. Neither of these taxonomies reflect the automated aspects of UE methods.

The sole existing survey of usability evaluation automation, by Balbo [1995], uses a taxonomy which distinguishes among four approaches to automation:

- **Non Automatic**: methods "performed by human factors specialists."
- **Automatic Capture**: methods that "rely on software facilities to record relevant information about the user and the system, such as visual data, speech acts, and keyboard and mouse actions."
- **Automatic Analysis**: methods that are "able to identify usability problems automatically."
- **Automatic Critic**: methods which "not only point out difficulties but propose improvements."

Balbo uses these categories to classify thirteen common and uncommon UE methods. However, most of the methods surveyed require extensive human effort, because they rely on formal usability testing and/or require extensive evaluator interaction. For example, Balbo classifies several techniques for processing log files as automatic analysis methods despite the fact that these approaches require formal testing or informal use to generate those log files. What Balbo calls an automatic critic method may require the evaluator to create a complex UI model as input. Thus, this classification scheme is somewhat misleading since it ignores the non-automated requirements of the UE methods.

2.3.1 Proposed Taxonomy

To facilitate discussion of usability evaluation methods, UE methods are grouped along the following four dimensions:

- **Method Class:** describes the type of evaluation conducted at a high level (e.g., usability testing or simulation);
- **Method Type:** describes *how* the evaluation is conducted within a method class, such as thinking-aloud protocol (usability testing class) or information processor modeling (simulation class);
- **Automation Type:** describes the evaluation aspect that is automated (e.g., capture, analysis, or critique); and
- **Effort Level:** describes the type of effort required to execute the method (e.g., model development or interface usage).

Method Class

UE methods are classified into five method classes: testing, inspection, inquiry, analytical modeling, and simulation.

- **Testing:** an evaluator observes participants interacting with an interface (i.e., completing tasks) to determine usability problems.

- **Inspection:** an evaluator uses a set of criteria or heuristics to identify potential usability problems in an interface.
- **Inquiry:** participants provide feedback on an interface via interviews, surveys, etc.
- **Analytical Modeling:** an evaluator employs user and interface models to generate usability predictions.
- **Simulation:** an evaluator employs user and interface models to mimic a user interacting with an interface and report the results of this interaction (e.g., simulated activities, errors, and other quantitative measures).

UE methods in the testing, inspection, and inquiry classes are appropriate for formative (i.e., identifying specific usability problems) and summative (i.e., obtaining general assessments of usability) purposes. Analytical modeling and simulation are engineering approaches to UE that enable evaluators to predict usability with user and interface models. Software engineering practices have had a major influence on the first three classes, while the latter two, analytical modeling and simulation, are quite similar to performance evaluation techniques used to analyze the performance of computer systems [Jain 1991]. Chapter 3 discusses performance evaluation techniques in detail.

Method Type

There are a wide range of evaluation methods within the testing, inspection, inquiry, analytical modeling, and simulation classes. Rather than discuss each method individually, one or more related methods are grouped into method types; this type typically describes *how* an evaluation is performed. Sections 2.5 – 2.9 present method types.

Automation Type

Balbo's automation taxonomy (described above) was adapted to specify which aspect of a usability evaluation method is automated: none, capture, analysis or critique.

- **None:** no level of automation supported (i.e., evaluator performs all aspects of the evaluation method).
- **Capture:** software automatically records usability data (e.g., logging interface usage).
- **Analysis:** software automatically identifies potential usability problems.
- **Critique:** software automates analysis and suggests improvements.

Effort Level

Balbo's automation taxonomy is also expanded to include consideration of a method's non-automated requirements. Each UE method is augmented with an attribute called **effort level**; this indicates the human effort required for method execution:

- **Minimal Effort:** does not require interface usage or modeling.
- **Model Development:** requires the evaluator to develop a UI model and/or a user model to employ the method.

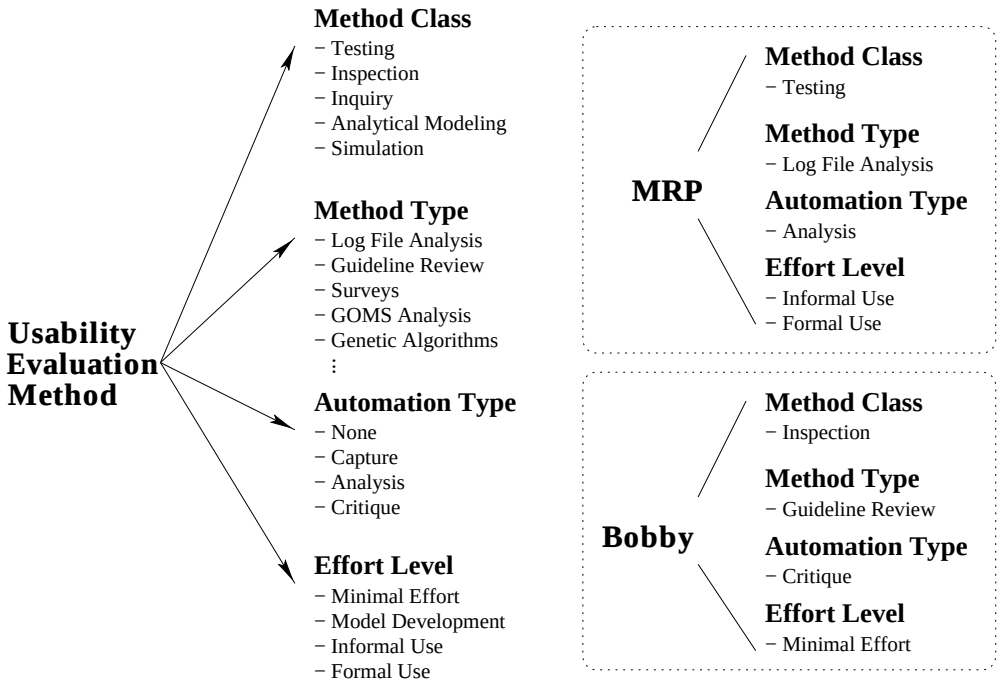


Figure 2.2: Summary of the proposed taxonomy for classifying usability evaluation methods. The right side of the figure demonstrates the taxonomy with two evaluation methods that will be discussed in later sections.

- **Informal Use:** requires completion of freely chosen tasks (i.e., unconstrained use by a user or evaluator).
- **Formal Use:** requires completion of specially selected tasks (i.e., constrained use by a user or evaluator).

These levels are not necessarily ordered by the amount of effort required, since this depends on the method used.

Summary

Figure 2.2 provides a synopsis of the proposed taxonomy and demonstrates it with two evaluation methods. The taxonomy consists of: a method class (testing, inspection, inquiry, analytical modeling, and simulation); a method type (e.g., log file analysis, guideline review, surveys, etc.); an automation type (none, capture, analysis, and critique); and an effort level (minimal, model, informal, and formal). This taxonomy is used to analyze evaluation methods in the remainder of this chapter.

2.4 Overview of Usability Evaluation Methods

Seventy-five UE methods applied to WIMP interfaces and fifty-eight methods applied to Web UIs were surveyed and analyzed using the proposed taxonomy. Of these 133 methods, only 29 apply to both Web and WIMP UIs. The applicability of each method was determined based on

the types of interfaces a method was used to evaluate in the literature and the author's judgment of whether or not the method could be used with other types of interfaces. Tables 2.1 and 2.2 combine survey results for both types of interfaces showing method classes (bold entries in the first column) and method types within each class (entries that are not bold in the first column). Each entry in columns two through five depicts specific UE methods along with the automation support available and the human effort required to employ automation. For some UE methods, more than one approach will be discussed; hence, the number of methods surveyed is shown in parenthesis beside the effort level.

Some approaches provide automation support for multiple method types (see Appendix A). Hence, Tables 2.1 and 2.2 contain only 111 methods. Tables 2.1 and 2.2 also depict methods applicable to both WIMP and Web UIs only once. Table 2.3 provides descriptions of all method types.

There are major differences in automation support among the five method classes. Overall, automation patterns are similar for WIMP and Web interfaces, with the exception that analytical modeling and simulation are far less explored in the Web domain than for WIMP interfaces (two vs. sixteen methods). Appendix A shows the information in Tables 2.1 and 2.2 separated by UI type. Table 2.4 summarizes the number of non-automated and automated capture, analysis, and critique methods surveyed overall and for each type of interface.

Tables 2.1 and 2.2 show that automation in general is greatly underexplored. Table 2.4 shows that methods without automation support represent 64% of the methods surveyed, while methods with automation support collectively represent only 36%. Of this 36%, capture methods represent 15%, analysis methods represent 19% and critique methods represent 2%. All but two of the capture methods require some level of interface usage; genetic algorithms and information scent modeling both use simulation to generate usage data for subsequent analysis. Of all of the surveyed methods, only 29% are free from requirements of formal or informal interface use.

To provide the fullest automation support, software would have to critique interfaces without requiring formal or informal use. The survey revealed that this level of automation has been developed for only one method type: guideline review (e.g., [Farenc and Palanque 1999; Lowgren and Nordqvist 1992; Scholtz and Laskowski 1998]). Guideline review methods automatically detect and report usability violations and then make suggestions for fixing them (discussed further in Section 2.6).

Of those methods that support the next level of automation – analysis – Tables 2.1 and 2.2 show that analytical modeling and simulation methods represent the majority. Most of these methods do not require formal or informal interface use.

The next sections discuss the various UE methods and their automation in more detail. Some methods are applicable to both WIMP and Web interfaces; however, distinctions are made where necessary about a method's applicability. The discussion also presents assessments of automated capture, analysis, and critique techniques using the following criteria:

- **Effectiveness:** how well a method discovers usability problems,
- **Ease of use:** how easy is a method to employ,
- **Ease of learning:** how easy is a method to learn, and
- **Applicability:** how widely applicable is a method to WIMP and/or Web UIs other than those originally applied to.

The effectiveness, ease of use, ease of learning, and applicability of automated methods is highlighted in discussions of each method class.

Method Class / Method Type	**Automation Type**			
	None	**Capture**	**Analysis**	**Critique**
Testing				
Thinking-aloud Protocol	F (1)			
Question-asking Protocol	F (1)			
Shadowing Method	F (1)			
Coaching Method	F (1)			
Teaching Method	F (1)			
Co-discovery Learning	F (1)			
Performance Measurement	F (1)	F (8)		
Log File Analysis			IFM (20)*	
Retrospective Testing	F (1)			
Remote Testing		IF (3)		
Inspection				
Guideline Review	IF (6)		(8)	M (11)†
Cognitive Walkthrough	IF (2)	F (1)		
Pluralistic Walkthrough	IF (1)			
Heuristic Evaluation	IF (1)			
Perspective-based Inspection	IF (1)			
Feature Inspection	IF (1)			
Formal Usability Inspection	F (1)			
Consistency Inspection	IF (1)			
Standards Inspection	IF (1)			
Inquiry				
Contextual Inquiry	IF (1)			
Field Observation	IF (1)			
Focus Groups	IF (1)			
Interviews	IF (1)			
Surveys	IF (1)			
Questionnaires	IF (1)	IF (2)		
Self-reporting Logs	IF (1)			
Screen Snapshots	IF (1)			
User Feedback	IF (1)			

Table 2.1: Automation support for WIMP and Web UE methods (Table 1 of 2). A number in parentheses indicates the number of UE methods surveyed for a particular method type and automation type. The effort level for each method is represented as: minimal (blank), formal (F), informal (I) and model (M). The * for the IFM entry indicates that either formal or informal interface use is required. In addition, a model may be used in the analysis. The † indicates that methods may or may not require a model.

Method Class / Method Type	Automation Type: None	Capture	Analsis	Critique
Analytical Modeling				
GOMS Analysis	M (4)		M (2)	
UIDE Analysis			M (2)	
Cognitive Task Analysis			M (1)	
Task-environment Analysis	M (1)			
Knowledge Analysis	M (2)			
Design Analysis	M (2)			
Programmable User Models			M (1)	
Simulation				
Information Proc. Modeling			M (9)	
Petri Net Modeling			FM (1)	
Genetic Algorithm Modeling		(1)		
Information Scent Modeling		M (1)		

Table 2.2: Automation support for WIMP and Web UE methods (Table 2 of 2). A number in parentheses indicates the number of UE methods surveyed for a particular method type and automation type. The effort level for each method is represented as: minimal (blank), formal (F), informal (I) and model (M).

2.5 Usability Testing Methods

Usability testing with real participants is a fundamental evaluation method [Nielsen 1993; Shneiderman 1998]. It provides an evaluator with direct information about how people use computers and what some of the problems are with the interface being tested. During usability testing, participants use the system or a prototype to complete a pre-determined set of tasks while the tester or software records the results of the participants' work. The tester then uses these results to determine how well the interface supports users' task completion and to derive other measures, such as the number of errors and task completion time.

Automation has been used predominantly in two ways within usability testing: automated capture of use data and automated analysis of this data according to some metrics or a model (referred to as log file analysis in Table 2.1). In rare cases methods support both automated capture and analysis of usage data [Al-Qaimari and McRostie 1999; Hong *et al.* 2001; Uehling and Wolf 1995].

2.5.1 Usability Testing Methods: Non-automated

This section provides a synopsis of eight non-automated method types. All method types have been or could be applied to both WIMP and Web UIs and require formal interface usage. Two testing protocols, thinking-aloud and question-asking, and six non-automated method types were surveyed; the testing protocols can be used with the other six method types in most cases. The major differences among the testing method types are the actual testing procedure (e.g., participants are silent, think aloud, or have the ability to ask an expert questions) and the intended outcome of testing (e.g., an understanding of the participant's mental model of the system or quantitative data).

Method types are summarized below. The method type (i.e., how an evaluation is performed) and the method (i.e., specific instantiation of a method type) are the same in all cases. Unless otherwise noted, most discussions are based on [Dix *et al.* 1998; Hom 1998; Human Factors Engineering 1999b; Nielsen 1993; Shneiderman 1998].

Method Class Method Type	**Description**
Testing	
Thinking-aloud Protocol	user talks during test
Question-asking Protocol	tester asks user questions
Shadowing Method	expert explains user actions to tester
Coaching Method	user can ask an expert questions
Teaching Method	expert user teaches novice user
Co-discovery Learning	two users collaborate
Performance Measurement	tester or software records usage data during test
Log File Analysis	tester analyzes usage data
Retrospective Testing	tester reviews videotape with user
Remote Testing	tester and user are not co-located during test
Inspection	
Guideline Review	expert checks guideline conformance
Cognitive Walkthrough	expert simulates user's problem solving
Pluralistic Walkthrough	multiple people conduct cognitive walkthrough
Heuristic Evaluation	expert identifies heuristic violations
Perspective-based Inspection	expert conducts narrowly focused heuristic evaluation
Feature Inspection	expert evaluates product features
Formal Usability Inspection	experts conduct formal heuristic evaluation
Consistency Inspection	expert checks consistency across products
Standards Inspection	expert checks for standards compliance
Inquiry	
Contextual Inquiry	interviewer questions users in their environment
Field Observation	interviewer observes system use in user's environment
Focus Groups	multiple users participate in a discussion session
Interviews	one user participates in a discussion session
Surveys	interviewer asks user specific questions
Questionnaires	user provides answers to specific questions
Self-reporting Logs	user records UI operations
Screen Snapshots	user captures UI screens
User Feedback	user submits comments
Analytical Modeling	
GOMS Analysis	predict execution and learning time
UIDE Analysis	conduct GOMS analysis within a UIDE
Cognitive Task Analysis	predict usability problems
Task-environment Analysis	assess mapping of user's goals into UI tasks
Knowledge Analysis	predict learnability
Design Analysis	assess design complexity
Programmable User Models	write program that acts like a user
Simulation	
Information Proc. Modeling	mimic user interaction
Petri Net Modeling	mimic user interaction from usage data
Genetic Algorithm Modeling	mimic novice user interaction
Information Scent Modeling	mimic Web site navigation

Table 2.3: Descriptions of the WIMP and Web UE method types depicted in Table 2.1.

Methods Surveyed	Automation Type			
	None	Capture	Analysis	Critique
Overall				
Total	30	7	9	1
Percent	64%	15%	19%	2%
WIMP UIs				
Total	30	5	8	1
Percent	68%	11%	18%	2%
Web UIs				
Total	26	5	4	1
Percent	72%	14%	11%	3%

Table 2.4: Summary of UE methods surveyed for each automation type.

Thinking-aloud Protocol. The thinking-aloud protocol requires participants to verbalize their thoughts, feelings, and opinions during a usability test. One goal of this approach is to enable the tester to get a better understanding of the participant's mental model during interaction with the interface. Critical response and periodic report are two variations of the protocol wherein the participant is vocal only during the execution of certain pre-determined tasks or at pre-determined intervals of time, respectively.

Question-asking Protocol. This method is an extension of the thinking-aloud protocol wherein testers prompt participants by asking direct questions about the interface. The goal of such questioning is to enable the tester to get an even better understanding of the participant's mental model of the system.

Coaching Method. The coaching method allows participants to ask any system-related questions of an expert coach during usability testing. Usually, the tester acts as the expert coach, but it is possible to have a separate tester serving as a coach. The latter approach may allow the tester to gain additional usability insight through observing the interaction between the participant and coach. In cases where an expert user serves as the coach, this also enables the tester to analyze the expert user's mental model of the system. The main goal of this method is to determine the information needs of users to provide better training and documentation in addition to possibly redesigning the interface to eliminate the need for questions in the first place. It is also possible for the tester to control the answers given to questions during testing to discover what types of answers help users the most.

Teaching Method. For this method, the participant interacts with the system first to develop expertise to subsequently teach a novice user about the system. The novice user serves as a student and does not actively engage in problem solving. The participant does the problem solving, explains to the novice user how the system works, and demonstrates a set of pre-determined tasks. This method enables testers to assess the ease of learning of an interface.

Shadowing Method. Shadowing is an alternative to the thinking-aloud protocol wherein an expert user sits next to the tester and explains the participant's behavior during the testing session. Evaluators use this method in situations where it is inappropriate for participants to think aloud or talk to the tester (e.g., collecting performance measurements).

Co-discovery Learning. During a co-discovery learning session, two participants attempt to perform the tasks together while the tester observes their interaction. This approach is similar

to the type of collaboration that occurs naturally in other environments, such as at work. As the participants complete tasks, the tester encourages them to explain what they are thinking about in a manner similar to the thinking-aloud protocol.

Performance Measurement. The goal of this testing method is to capture quantitative data about participants' performance when they complete tasks. As such, there is usually no interaction between the tester and participant during the test. Evaluators usually conduct such testing in a usability lab to facilitate accurate data collection and to minimize interference. Sometimes this method is combined with other techniques to capture qualitative data as well, such as retrospective testing (discussed immediately below).

Measurement studies form the foundation of usability testing, since evaluators can use the results to assess whether the usability goals have been met as well as for competitive analysis. In the first case the evaluator would re-define an abstract performance goal, such as usability, into a specific usability attribute, such as efficiency of use. After specifying a specific usability attribute, the evaluator can quantify this attribute with a metric (e.g., time to complete a task, time spent recovering from errors, etc.) and devise a plan for measuring this metric in the interface and collecting the necessary performance data. Without automated tools, this collection is typically accomplished by taking notes or video taping testing sessions and subsequently reviewing the videotape to compute performance measures.

MUSiC (Metrics for Usability Standards in Computing) [Bevan and Macleod 1994; Macleod *et al.* 1997] is a rigorous performance measurement method developed by a consortium of European institutions, including Serco Usability Services (formerly the National Physical Laboratory), the University College Cork, and the HUSAT (Human Sciences and Advanced Technology) Institute. Applying the methodology entails: conducting a formal usability context analysis (i.e., determining who the users are, how they use the UI, and in what situations they use it) and following the performance measurement method [Rengger *et al.* 1993] as prescribed. MUSiC includes tools to support automated analysis of video recording using DRUM (Diagnostic Recorder for Usability Measurement, discussed in Section 2.5.2) [Macleod and Rengger 1993] as well as collecting subjective usability data via the SUMI (Software Usability Measurement Inventory, discussed in Section 2.7.2) [Porteous *et al.* 1993] questionnaire.

Retrospective Testing. This method is a followup to any other videotaped testing session wherein the tester and participant review the videotape together. During this review, the tester asks the participant questions regarding her behavior during the test. The goal of this review is to collect additional information from the usability test. Although such testing can be valuable, it substantially increases the cost of usability testing because each test takes at least twice as long to conduct.

2.5.2 Usability Testing Methods: Automated Capture

Many usability testing methods require the recording of actions a user makes while exercising an interface. This can be done by an evaluator taking notes while the participant uses the system, either live or by repeatedly viewing a videotape of the session; both are time-consuming activities. As an alternative, automated capture techniques can log user activity automatically. An important distinction can be made between information that is easy to record but difficult to interpret, such as keystrokes, and information that is meaningful but difficult to automatically

Method Class: Testing		
Automation Type: Capture		
Method Type: Performance Measurement - software records usage data during test (8 methods)		
UE Method	**UI**	**Effort**
Log low-level events ([Hammontree *et al.* 1992])	WIMP	F
Log UIMS events (UsAGE, IDCAT)	WIMP	F
Log system-level events (KALDI)	WIMP	F
Log Web server requests ([Scholtz and Laskowski 1998])	Web	F
Log client-side activities (WebVIP, WET)	Web	F
Log Web proxy requests (WebQuilt)	Web	F
Method Type: Remote Testing - tester and user are not co-located (3 methods)		
UE Method	**UI**	**Effort**
Employ same-time different-place testing (KALDI)	WIMP, Web	IF
Employ different-time different-place testing (journaled sessions)	WIMP, Web	IF
Analyze a Web site's information organization (WebCAT)	Web	IF

Table 2.5: Synopsis of automated capture support for usability testing methods.

label, such as task completion. Automated capture approaches vary with respect to the granularity of information captured.

Within the usability testing class of UE, automated capture of usage data is supported by two method types: performance measurement and remote testing. Both require the instrumentation of a user interface, incorporation into a user interface management system (UIMS), or capture at the system level. A UIMS [Olsen, Jr. 1992] is a software library that provides high-level abstractions for specifying portable and consistent interface models that are then compiled into UI implementations on each platform similarly to Java programs. Table 2.5 provides a synopsis of automated capture methods discussed in the remainder of this section. Support available for WIMP and Web UIs is discussed separately.

Usability Testing Methods: Automated Capture – WIMP UIs

Performance measurement methods record usage data (e.g., a log of events and times when events occurred) during a usability test. Video recording and event logging tools [Al-Qaimari and McRostie 1999; Hammontree *et al.* 1992; Uehling and Wolf 1995] are available to automatically and accurately align timing data with user interface events. Some event logging tools (e.g., [Hammontree *et al.* 1992]) record events at the keystroke or system level. Recording data at this level produces voluminous log files and makes it difficult to map recorded usage into high-level tasks.

As an alternative, two systems log events within a UIMS. UsAGE (User Action Graphing Effort)[4] [Uehling and Wolf 1995] enables the evaluator to replay logged events, meaning it can replicate logged events during playback. This requires that the same study data (databases, documents, etc.) be available during playback as was used during the usability test. IDCAT (Integrated Data Capture and Analysis Tool) [Hammontree *et al.* 1992] logs events and automatically filters and classifies them into meaningful actions. This system requires a video recorder to synchronize taped footage with logged events. KALDI (Keyboard/mouse Action Logger and Display Instrument) [Al-Qaimari and McRostie 1999] supports event logging and screen capturing via Java and

[4]This method is not to be confused with the USAGE analytical modeling approach discussed in Section 2.8.

does not require special equipment. Both KALDI and UsAGE also support log file analysis (see Section 2.5.3).

Remote testing methods enable testing between a tester and participant who are not co-located. In this case the evaluator is not able to observe the user directly, but can gather data about the process over a computer network. Remote testing methods are distinguished according to whether or not a tester observes the participant during testing or not. Same-time different-place and different-time different-place are two major remote testing approaches [Hartson *et al.* 1996].

In same-time different-place or remote-control testing the tester observes the participant's screen through network transmissions (e.g., using PC Anywhere or Timbuktu) and may be able to hear what the participant says via a speaker telephone or a microphone affixed to the computer. Software makes it possible for the tester to interact with the participant during the test, which is essential for techniques like the question-asking or thinking aloud protocols that require such interaction.

The tester does not observe the user during different-time different-place testing. An example of this approach is the journaled session [Nielsen 1993], in which software guides the participant through a testing session and logs the results. Evaluators can use this approach with prototypes to get feedback early in the design process, as well as with released products. In the early stages, evaluators distribute disks containing a prototype of a software product and embedded code for recording users' actions. Users experiment with the prototype and return the disks to evaluators upon completion. It is also possible to embed dialog boxes within the prototype to record users' comments or observations during usage. For released products, evaluators use this method to capture statistics about the frequency with which the user has used a feature or the occurrence of events of interest (e.g., error messages). This information is valuable for optimizing frequently-used features and the overall usability of future releases.

Remote testing approaches allow for wider testing than traditional methods, but evaluators may experience technical difficulties with hardware and/or software components (e.g., inability to correctly configure monitoring software or network failures). This can be especially problematic for same-time different-place testing where the tester needs to observe the participant during testing. Most techniques also have restrictions on the types of UIs to which they can be applied. This is mainly determined by the underlying hardware (e.g., PC Anywhere only operates on PC platforms) [Hartson *et al.* 1996]. KALDI, mentioned above, also supports remote testing. Since it was developed in Java, evaluators can use it for same- and different-time testing of Java applications on a wide range of computing platforms.

Usability Testing Methods: Automated Capture – Web UIs

The Web enables remote testing and performance measurement on a much larger scale than is feasible with WIMP interfaces. Both same-time different-place and different-time different-place approaches can be employed for remote testing of Web UIs. Similar to journaled sessions, Web servers maintain usage logs and automatically generate a log file entry for each request. These entries include the IP address of the requester, request time, name of the requested Web page, and in some cases the URL of the referring page (i.e., where the user came from). Server logs cannot record user interactions that occur only on the client side (e.g., use of within-page anchor links or back button), and the validity of server log data is questionable due to caching by proxy servers and browsers [Etgen and Cantor 1999; Scholtz and Laskowski 1998]. Server logs may not reflect usability, especially since these logs are often difficult to interpret [Schwartz 2000] and users' tasks may not be discernible [Byrne *et al.* 1999; Schwartz 2000].

Client-side logs capture more accurate, comprehensive usage data than server-side logs

because they allow all browser events to be recorded. Such logging may provide more insight about usability. On the downside, it requires every Web page to be modified to log usage data, or else use of an instrumented browser or special proxy server.

The NIST WebMetrics tool suite [Scholtz and Laskowski 1998] captures client-side usage data. This suite includes WebVIP (Web Visual Instrumentor Program), a visual tool that enables the evaluator to add event handling code to Web pages. This code automatically records the page identifier and a time stamp in an ASCII file every time a user selects a link. (This package also includes a visualization tool, VISVIP [Cugini and Scholtz 1999], for viewing logs collected with WebVIP; see Section 2.5.3.) Using this client-side data, the evaluator can accurately measure time spent on tasks or particular pages as well as study use of the back button and user clickstreams. Despite its advantages over server-side logging, WebVIP requires the evaluator to make a copy of an entire Web site, which could lead to invalid path specifications and other difficulties with getting the copied site to function properly. The evaluator must also add logging code to each individual link on a page. Since WebVIP only collects data on selected HTML links, it does not record interactions with other Web objects, such as forms. It also does not record usage of external or non-instrumented links.

Similar to WebVIP, the Web Event-logging Tool (WET) [Etgen and Cantor 1999] supports the capture of client-side data, including clicks on Web objects, window resizing, typing in a form object and form resetting. WET interacts with Microsoft Internet Explorer and Netscape Navigator to record browser event information, including the type of event, a time stamp, and the document-window location. This gives the evaluator a more complete view of the user's interaction with a Web interface than WebVIP. WET does not require as much effort to employ as WebVIP, nor does it suffer from the same limitations. To use this tool, the evaluator specifies events (e.g., clicks, changes, loads, and mouseovers) and event handling functions in a text file on the Web server; sample files are available to simplify this step. The evaluator must also add a single call to the text file within the <head> tag of each Web page to be logged. Currently, the log file analysis for both WebVIP and WET is manual. Future work has been proposed to automate this analysis.

As an alternative to server-side and client-side logging, WebQuilt [Hong *et al.* 2001] uses proxy-based logging to capture usage data. The system automatically captures Web server requests using a special proxy server, logs requests, and subsequently routes requests to the Web server. All of the links in Web pages are also redirected to the proxy server; this eliminates the need for users to manually configure their browsers to route requests to the proxy. The system captures more accurate site usage details (e.g., use of the back button) than server-side logging, makes it possible to run usability tests on any Web site (e.g., for competitive analysis), and makes it possible to track participants accurately, since extra information can be encoded in the study URL. Although the system does not capture many client-side details, such as the use of page elements or window resizing, it does simplify instrumenting a site for logging, since this is done automatically. The WebQuilt system also supports task-based analysis and visualization of captured usage data (see Section 2.5.3).

The NIST WebMetrics tool suite also includes WebCAT (Category Analysis Tool), a tool that aids in Web site category analysis, by a technique sometimes known as card sorting [Nielsen 1993]. In non-automated card sorting, the evaluator (or a team of evaluators) writes concepts on pieces of paper, and users group the topics into piles. The evaluator manually analyzes these groupings to determine a good category structure. WebCAT allows the evaluator to test proposed topic categories for a site via a category matching task (a variation of card-sorting where users assign concepts to predefined categories); this task can be completed remotely by users. Results are compared to the designer's category structure, and the evaluator can use the analysis to inform the best information organization for a site. WebCAT enables wider testing and faster analysis than

traditional card sorting, and helps make the technique scale for a large number of topic categories.

Usability Testing Methods: Automated Capture – Discussion

Automated capture methods represent important first steps toward informing UI improvements – they provide input data for analysis and in the case of remote testing, enable the evaluator to collect data for a larger number of users than traditional methods. Without this automation, evaluators would have to manually record usage data, expend considerable time reviewing videotaped testing sessions or in the case of the Web, rely on questionable server logs. Methods such as KALDI and WET capture high-level events that correspond to specific tasks or UI features. KALDI also supports automated analysis of captured data as discussed below.

Table 2.5 summarizes performance measurement and remote testing methods discussed in this section. It is difficult to assess the ease of use and learning of these approaches, especially IDCAT and remote testing approaches that require integration of hardware and software components, such as video recorders and logging software. For client-side Web site logging, WET appears to be easier to use and learn than WebVIP. It requires the creation of an event handling file and the addition of a small block of code in each Web page header, while WebVIP requires the evaluator to add code to every link on all Web pages. WET also enables the evaluator to capture more comprehensive usage data than WebVIP. WebQuilt is easier to use and learn than WET because a proxy server automatically instruments a site for logging; however, it does not capture the same level of detail as client-side logging. WebCAT appears straightforward to use and learn for topic category analysis. Both remote testing and performance measurement techniques have restrictions on the types of UIs to which they can be applied. This is mainly determined by the underlying hardware (e.g., PC Anywhere only operates on PC platforms) or UIMS, although KALDI can potentially be used to evaluate Java applications on a wide range of platforms.

2.5.3 Usability Testing Methods: Automated Analysis

Log file analysis methods automate the analysis of data captured during formal or informal interface use. Since Web servers automatically log client requests, log file analysis is a heavily used methodology for evaluating Web interfaces [Drott 1998; Fuller and de Graaff 1996; Hochheiser and Shneiderman 2001; Sullivan 1997]. The survey revealed four general approaches for analyzing WIMP and Web log files: metric-based, pattern-matching, task-based, and inferential. Table 2.6 provides a synopsis of automated analysis methods discussed in the remainder of this section. Support available for the four general approaches is discussed separately.

Usability Testing Methods: Automated Analysis – Metric-Based Analysis of Log Files

Metric-based approaches generate quantitative performance measurements. Three examples for WIMP interfaces are DRUM [Macleod and Rengger 1993], the MIKE UIMS [Olsen, Jr. and Halversen 1988], and AMME (Automatic Mental Model Evaluator) [Rauterberg 1995; Rauterberg 1996b; Rauterberg and Aeppili 1995]. DRUM enables the evaluator to review a video tape of a usability test and manually log starting and ending points for tasks. DRUM processes this log and derives several measurements, including: task completion time, user efficiency (i.e., effectiveness divided by task completion time), and productive period (i.e., portion of time the user did not have problems). DRUM also synchronizes the occurrence of events in the log with videotaped footage, thus speeding up video analysis.

The MIKE UIMS enables an evaluator to assess the usability of a UI specified as a model that can be rapidly changed and compiled into a functional UI. MIKE captures usage data and

Method Class: Testing		
Automation Type: Analysis		
Method Type: Log File Analysis - analyze usage data (20 methods)		
UE Method	**UI**	**Effort**
Use metrics during log file analysis (DRUM, MIKE UIMS, AMME)	WIMP	IF
Use metrics during log file analysis (Service Metrics, [Bacheldor 1999])	Web	IF
Use pattern matching during log file analysis (MRP)	WIMP	IF
Use task models during log file analysis (IBOT, QUIP, WebQuilt, KALDI, UsAGE)	WIMP	IF
Use task models and pattern matching during log file analysis (ÉMA, USINE, RemUSINE)	WIMP	IFM
Visualization of log files ([Guzdial *et al.* 1994])	WIMP	IF
Statistical analysis or visualization of log files (traffic- and time-based analyses, VISVIP, Starfield and Dome Tree visualizations)	Web	IF

Table 2.6: Synopsis of automated analysis support for usability testing methods.

generates a number of general, physical, logical, and visual metrics, including performance time, command frequency, the number of physical operations required to complete a task, and required changes in the user's focus of attention on the screen. MIKE also calculates these metrics separately for command selection (e.g., traversing a menu, typing a command name, or hitting a button) and command specification (e.g., entering arguments for a command) to help the evaluator locate specific problems within the UI.

AMME employs petri nets [Petri 1973] to reconstruct and analyze the user's problem solving process. It requires a specially-formatted log file and a manually-created system description file (i.e., a list of interface states and a state transition matrix) to generate the petri net. It then computes measures of behavioral complexity (i.e., steps taken to perform tasks), routinization (i.e., repetitive use of task sequences), and ratios of thinking vs. waiting time. User studies with novices and experts validated these quantitative measures and showed behavioral complexity to correlate negatively with learning (i.e., less steps are taken to solve tasks as a user learns the interface) [Rauterberg and Aeppili 1995]. Hence, the behavioral complexity measure provides insight on interface complexity. It is also possible to simulate the generated petri net (see Section 2.9) to further analyze the user's problem solving and learning processes. Multidimensional scaling and Markov analysis tools are available for comparing multiple petri nets (e.g., nets generated from novice and expert user logs). Since AMME processes log files, it could easily be extended to Web interfaces.

For the Web, site analysis tools developed by Service Metrics [Service Metrics 1999] and others [Bacheldor 1999] allow evaluators to pinpoint performance bottlenecks, such as slow server response time, that may negatively impact the usability of a Web site. Service Metrics' tools, for example, can collect performance measures from multiple geographical locations under various access conditions. In general, performance measurement approaches focus on server and network performance, but provide little insight into the usability of the Web site itself.

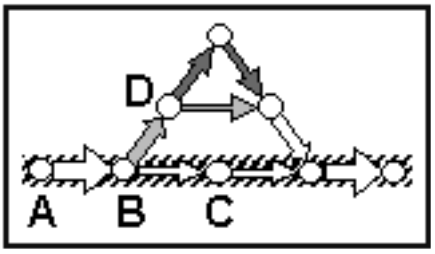


Figure 2.3: QUIP usage profile contrasting task flows for two users to the designer's task flow (diagonal shading) [Helfrich and Landay 1999]. Each node represents a user action, and arrows indicate actions taken by users. The width of arrows denotes the fraction of users completing actions, while the color of arrows reflects the average time between actions (darker colors correspond to longer time). Reprinted with permission of the authors.

Usability Testing Methods: Automated Analysis – Pattern-Matching Analysis of Log Files

Pattern-matching approaches, such as MRP (Maximum Repeating Pattern) [Siochi and Hix 1991], analyze user behavior captured in logs. MRP detects and reports repeated user actions (e.g., consecutive invocations of the same command and errors) that may indicate usability problems. Studies with MRP showed the technique to be useful for detecting problems with expert users, but additional data prefiltering was required for detecting problems with novice users. Whether the evaluator performed this prefiltering or it was automated is unclear in the literature.

Three evaluation methods employ pattern matching in conjunction with task models. These methods are discussed immediately below.

Usability Testing Methods: Automated Analysis – Task-Based Analysis of Log Files

Task-based approaches analyze discrepancies between the designer's anticipation of the user's task model and what a user actually does while using the system. The IBOT system [Zettlemoyer *et al.* 1999] automatically analyzes log files to detect task completion events. The IBOT system interacts with Windows operating systems to capture low-level window events (e.g., keyboard and mouse actions) and screen buffer information (i.e., a screen image that can be processed to automatically identify widgets). The system then combines this information into interface abstractions (e.g., menu select and menubar search operations). Evaluators can use the system to compare user and designer behavior on these tasks and to recognize patterns of inefficient or incorrect behaviors during task completion. Without such a tool, the evaluator has to study the log files and do the comparison manually. Future work has been proposed to provide critique support.

The QUIP (Quantitative User Interface Profiling) tool [Helfrich and Landay 1999] and KALDI [Al-Qaimari and McRostie 1999] (see previous section) provide more advanced approaches to task-based, log file analysis for Java UIs. Unlike other approaches, QUIP aggregates traces of multiple user interactions and compares the task flows of these users to the designer's task flow. QUIP encodes quantitative time-based and trace-based information into directed graphs (see Figure 2.3). For example, the average time between actions is indicated by the color of each arrow, and the proportion of users who performed a particular sequence of actions is indicated by the width of each arrow. The designer's task flow is indicated by the diagonal shading in Figure 2.3. Currently, the evaluator must instrument the UI to collect the necessary usage data, and must manually analyze the graphs to identify usability problems.

WebQuilt [Hong *et al.* 2001] provides a graphical depiction of usage data captured via a

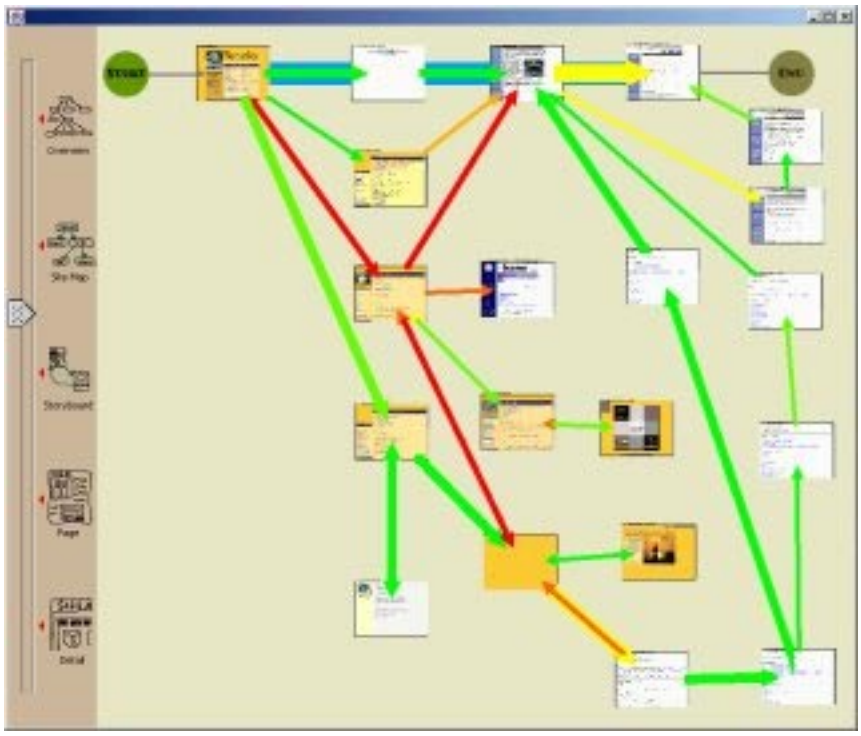


Figure 2.4: WebQuilt visualization contrasting task flows for twelve users to the designer's task flow (path across the top with thick shading) [Hong *et al.* 2001]. The circle on the left shows the start of the task, while the circle on the right shows the end of the task. Each thumbnail corresponds to a page in the Web site, and arrows indicate actions taken by users. The width of arrows denotes the fraction of users traversing the path, while the color of arrows reflects the average time users spent on pages before clicking a link (darker colors correspond to longer time). The interface enables evaluators to view usage data at different levels of detail. Reprinted with permission of the authors.

special proxy server. The visualization is very similar to QUIP, in that it aggregates usage data and summarizes task completion with arrows showing actions taken by users, the percentages of users taking the actions (width), and the average times users spent on pages before selecting links (color). WebQuilt also enables the evaluator to view the usage data at multiple levels of detail using a zooming interface. For example, if users spent a considerable amount of time on a Web page, the evaluator could view the actual page within the interface.

KALDI captures usage data and screen shots for Java applications. It also enables the evaluator to classify tasks (both manually and via automatic filters), compare user performance on tasks, and playback synchronized screen shots. It depicts logs graphically to facilitate analysis.

UsAGE [Uehling and Wolf 1995], which also supports logging usage data within a UIMS, provides a similar graphical presentation for comparing event logs for expert and novice users. However, graph nodes are labeled with UIMS event names, thus making it difficult to map events to specific interface tasks. To mitigate this shortcoming, UsAGE allows the evaluator to replay recorded events in the interface.

Several systems incorporate pattern-matching (see discussion above) into their analyses. This combination results in the most advanced log file analysis of all of the approaches surveyed. These systems include ÉMA (Automatic Analysis Mechanism for the Ergonomic Evaluation of User Interfaces) [Balbo 1996], USINE (USer Interface Evaluator) [Lecerof and Paternò 1998], and RemUSINE (Remote USer Interface Evaluator) [Paternò and Ballardin 1999], all discussed below.

ÉMA uses a manually-created data-flow task model and standard behavior heuristics to flag usage patterns that may indicate usability problems. ÉMA extends the MRP approach (repeated command execution) to detect additional patterns, including immediate task cancellation, shifts in direction during task completion, and discrepancies between task completion and the task model. ÉMA outputs results in an annotated log file, which the evaluator must manually inspect to identify usability problems. Application of this technique to the evaluation of ATM (Automated Teller Machine) usage corresponded with problems identified using standard heuristic evaluation [Balbo 1996].

USINE [Lecerof and Paternò 1998] employs the ConcurTaskTrees [Paternò *et al.* 1997] notation to express temporal relationships among UI tasks (e.g., enabling, disabling, and synchronization). Using this information, USINE looks for precondition errors (i.e., task sequences that violate temporal relationships) and also reports quantitative metrics (e.g., task completion time) and information about task patterns, missing tasks, and user preferences reflected in the usage data. Studies with a graphical interface showed that USINE's results correspond with empirical observations and highlight the source of some usability problems. To use the system, evaluators must create task models using the ConcurTaskTrees editor as well as a table specifying mappings between log entries and the task model. USINE processes log files and outputs detailed reports and graphs to highlight usability problems. RemUSINE [Paternò and Ballardin 1999] is an extension that analyzes multiple log files (typically captured remotely) to enable comparison across users.

Usability Testing Methods: Automated Analysis – Inferential Analysis of Log Files

Inferential analysis of Web log files includes both statistical and visualization techniques. Statistical approaches include traffic-based analysis (e.g., pages-per-visitor or visitors-per-page) and time-based analysis (e.g., click paths and page-view durations) [Drott 1998; Fuller and de Graaff 1996; Sullivan 1997; Theng and Marsden 1998]. Some methods require manual pre-processing or filtering of the logs before analysis. Furthermore, the evaluator must interpret reported measures to identify usability problems. Software tools, such as WebTrends [WebTrends Corporation 2000], facilitate analysis by presenting results in graphical and report formats.

Statistical analysis is largely inconclusive for Web server logs, since they provide only a partial trace of user behavior and timing estimates may be skewed by network latencies. Server log files are also missing valuable information about what tasks users want to accomplish [Byrne *et al.* 1999; Schwartz 2000]. Nonetheless, statistical analysis techniques have been useful for improving usability and enable ongoing, cost-effective evaluation throughout the life of a site [Fuller and de Graaff 1996; Sullivan 1997].

Visualization is also used for inferential analysis of WIMP and Web log files [Chi *et al.* 2000; Cugini and Scholtz 1999; Guzdial *et al.* 1994; Hochheiser and Shneiderman 2001]. It enables evaluators to filter, manipulate, and render log file data in a way that ideally facilitates analysis. [Guzdial *et al.* 1994] propose several techniques for analyzing WIMP log files, such as color coding patterns and command usage, tracking screen updates, and tracking mouseclick locations and depth (i.e., number of times the user clicked the mouse in screen areas). However, there is no discussion of how effective these approaches are in supporting analysis.

Starfield visualization [Hochheiser and Shneiderman 2001] is one approach that enables evaluators to interactively explore Web server log data to gain an understanding of human factors issues related to visitation patterns. This approach combines the simultaneous display of a large number of individual data points (e.g., URLs requested versus time of requests) in an interface that supports zooming, filtering, and dynamic querying [Ahlberg and Shneiderman 1994]. Visualizations provide a high-level view of usage patterns (e.g., usage frequency, correlated references, bandwidth

usage, HTTP errors, and patterns of repeated visits over time) that the evaluator must explore to identify usability problems. It would be beneficial to employ a statistical analysis approach to study traffic, clickstreams, and page views prior to exploring visualizations.

The Dome Tree visualization [Chi *et al.* 2000] provides an insightful representation of simulated (see Section 2.9) and actual Web usage captured in server log files. This approach maps a Web site into a three dimensional surface representing the hyperlinks (see top part of Figure 2.5). The location of links on the surface is determined by a combination of content similarity, link usage, and link structure of Web pages. The visualization highlights the most commonly traversed subpaths. An evaluator can explore these usage paths to possibly gain insight about the information "scent" (i.e., common topics among Web pages on the path) as depicted in the bottom window of Figure 2.5. This additional information may help the evaluator infer what the information needs of site users are, and more importantly, may help infer whether the site satisfies those needs. The Dome Tree visualization also reports a crude path traversal time based on the sizes of pages (i.e., number of bytes in HTML and image files) along the path. Server log accuracy limits the extent to which this approach can successfully indicate usability problems. As is the case for Starfield visualization, it would be beneficial to statistically analyze log files prior to using this approach.

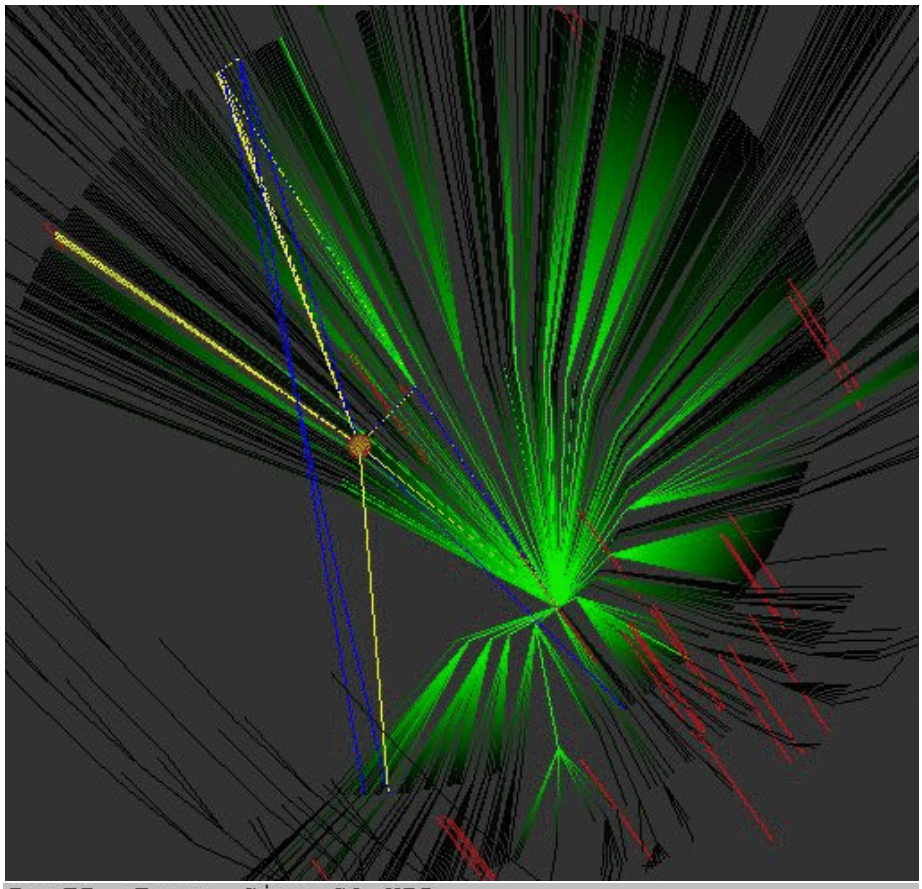
VISVIP [Cugini and Scholtz 1999] is a three-dimensional tool for visualizing log files compiled by WebVIP during usability testing (see previous section). Figure 2.6 shows VISVIP's Web site (top graph) and usage path (bottom graph) depictions to be similar to the Dome Tree visualization approach. VISVIP generates a 2D layout of the site where adjacent nodes are placed closer together than non-adjacent nodes. A third dimension reflects timing data as a dotted vertical bar at each node; the height is proportional to the amount of time. VISVIP also provides animation facilities for visualizing path traversal. Since WebVIP logs reflect actual task completion, prior statistical analysis is not necessary for VISVIP usage.

Usability Testing Methods: Automated Analysis – Discussion

Table 2.6 summarizes log file analysis methods discussed in this section. Although the techniques vary widely on the four assessment criteria (effectiveness, ease of use, ease of learning, and applicability), all approaches offer substantial benefits over the alternative – time-consuming, unaided analysis of potentially large amounts of raw data. Hybrid task-based pattern-matching techniques like USINE may be the most effective (i.e., provide clear insight for improving usability via task analysis), but they require additional effort and learning time over simpler pattern-matching approaches; this additional effort is mainly in the development of task models. Although pattern-matching approaches are easier to use and learn, they only detect problems for pre-specified usage patterns.

Metric-based approaches in the WIMP domain have been effective at associating measurements with specific interface aspects, such as commands and tasks, which can then be used to identify usability problems. AMME also helps the evaluator to understand the user's problem solving process and conduct simulation studies. However, metric-based approaches require the evaluator to conduct more analysis to ascertain the source of usability problems than task-based approaches. Metric-based techniques in the Web domain focus on server and network performance, which provides little usability insight. Similarly, inferential analysis of Web server logs is limited by their accuracy and may provide inconclusive usability information.

Most of the techniques surveyed in this section could be applied to WIMP and Web UIs other than those demonstrated on, with the exception of the MIKE UIMS and UsAGE, which require a WIMP UI to be developed within a special environment. AMME could be employed for both Web and WIMP UIs, provided log files and system models are available.



DocID	Freq	Size	SA	URL
33775	63	2445		docs/investor/sharinfo.htm
36369	168	3028		docs/factbook/1998/index.htm
33775	63	2445		docs/investor/sharinfo.htm
33778	58	20537		docs/investor/sitemap.htm
33780	26	2313		docs/investor/orderdoc.htm

Total Path Bytes: 30768 (17 Seconds)
Keywords for 33775 36369 33775 33778 33780:
reinvestment 7.230992
stock 6.651224
brochure 6.307321
dividend 5.522883
shareholder 5.165537
preferred 4.850075
automatic 4.215094
contacts 4.163801

Figure 2.5: Dome Tree visualization [Chi *et al.* 2000] of a Web site with a usage path displayed as a series of connected lines across the left side. The bottom part of the figure displays information about the usage path, including an estimated navigation time and information scent (i.e., common keywords along the path). Reprinted with permission of the authors.

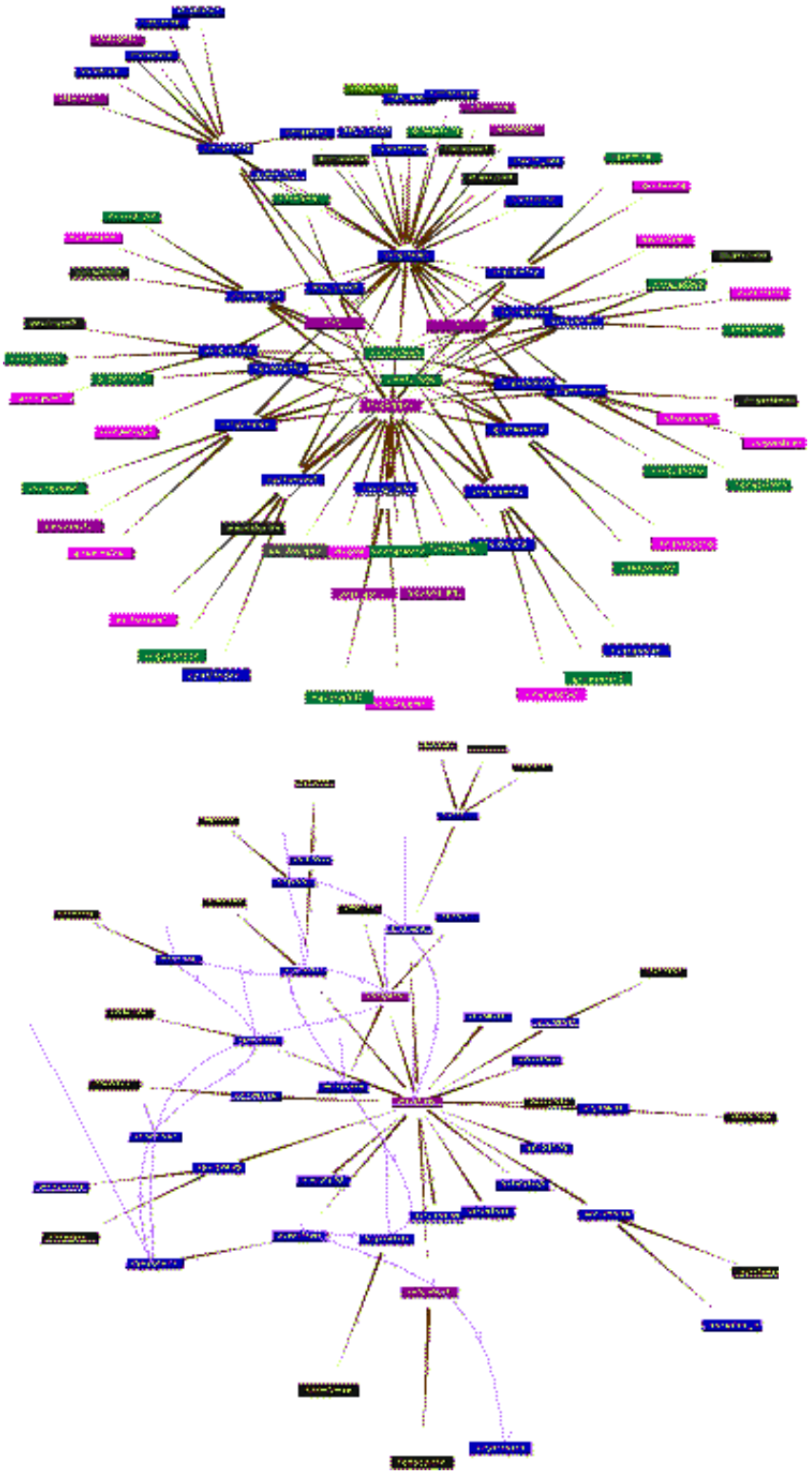


Figure 2.6: VISVIP visualization [Cugini and Scholtz 1999] of a Web site (top part). The bottom part of the figure displays a usage path (series of directed lines on the left site) laid over the site. Reprinted with permission of the authors.

2.6 Inspection Methods

A usability inspection is an evaluation methodology whereby an evaluator examines the usability aspects of a UI design with respect to its conformance to a set of guidelines. Guidelines can range from highly specific prescriptions to broad principles. Unlike other UE methods, inspections rely solely on the evaluator's judgment. A large number of detailed usability guidelines have been developed for WIMP interfaces [Open Software Foundation 1991; Smith and Mosier 1986] and Web interfaces [Comber 1995; Detweiler and Omanson 1996; Levine 1996; Lynch and Horton 1999; Web Accessibility Initiative 1999]. Common non-automated inspection techniques are heuristic evaluation [Nielsen 1993] and cognitive walkthroughs [Lewis *et al.* 1990].

Designers have historically experienced difficulties following design guidelines [Borges *et al.* 1996; de Souza and Bevan 1990; Lowgren and Nordqvist 1992; Smith 1986]. One study has demonstrated that designers are biased towards aesthetically pleasing interfaces, regardless of efficiency [Sears 1995]. Because designers have difficulty applying design guidelines, automation has been predominately used within the inspection class to objectively check guideline conformance. Software tools assist evaluators with guideline review by automatically detecting and reporting usability violations and in some cases making suggestions for fixing them [Balbo 1995; Farenc and Palanque 1999]. Automated capture, analysis, and critique support is available for the guideline review and cognitive walkthrough method types, as described in the remainder of this section.

2.6.1 Inspection Methods: Non-automated

Nine inspection method types were surveyed; they vary based on the goals of the inspection (e.g., guideline and standard conformance), the evaluative criteria (e.g., guidelines and standards), the evaluative process (e.g., formal or informal, task-based, or self-guided exploration), and how judgment is derived (e.g., individually or as a group). The fundamental goal of all inspection methods is to find usability problems in an existing interface design and then use these problems to make recommendations for improving the usability of an interface. Each inspection method has more specific objectives that aide in choosing the most appropriate method. For example, if the goal of a usability evaluation is to determine an interface's conformance to established guidelines, then the evaluator would use the guideline review method.

The nine inspection method types also differ in usability from the evaluator's perspective (i.e., how easy it is to learn and apply a method). Heuristic and perspective-based evaluations are considered to be easy to learn and apply, while cognitive walkthrough is not as easy to learn or apply [Nielsen and Mack 1994; Zhang *et al.* 1998]. Studies have also shown that the simpler the technique, the more effective the method is for identifying usability problems [Nielsen and Mack 1994]. For example, several studies have contrasted heuristic evaluation and cognitive walkthrough and reported heuristic evaluation to be more effective [Nielsen and Mack 1994].

In most cases, the method type is the same as the method. All of the methods require formal or informal interface usage and have been applied or could be applied to both WIMP and Web UIs. Unless otherwise specified, most discussions below are based on [Dix *et al.* 1998; Hom 1998; Human Factors Engineering 1999b; Nielsen 1993; Nielsen and Mack 1994; Shneiderman 1998].

Guideline Review. In guideline reviews evaluators check a WIMP interface for conformance with a comprehensive and sometimes large number (e.g., 1000 or more) of established usability guidelines. There are several accepted guidelines, including the Smith and Mosier guidelines [Smith and Mosier 1986] and the Motif style guides [Open Software Foundation 1991].

Guideline review is also actively used to evaluate Web interfaces. Many corporations have developed their own guidelines and there have been several efforts to develop a standard set of guidelines. Keevil [1998] presents a set of guidelines as a list of yes/no questions about the organization, user-oriented tasks, and technical content of a Web site. After answering these questions in a spreadsheet or Web form, a usability index can be computed for a site. Ambühler and Lindenmeyer [1999] use guidelines to compute accessibility measurements (i.e., how easy is it to access a page without special hardware or software). Lohse and Spiller [1998] use regression modeling on a set of 32 guidelines to predict store traffic and dollar sales as a function of interface features, such as the number of links into the store and number of products. Finally, Rossi *et al.* [1999] propose guidelines to assist designers with determining the best navigation structure for a site. Currently, all of these approaches require manual evaluation.

Ratner *et al.* [1996] question the validity of HTML usability guidelines, since most HTML guidelines have not been subjected to a rigorous development process as established guidelines for WIMP interfaces. Analysis of 21 HTML guidelines showed little consistency among them, with 75% of recommendations appearing in only one style guide. Furthermore, only 20% of HTML-relevant recommendations from established WIMP guidelines existed in the 21 HTML style guides.

Cognitive Walkthrough. Cognitive walkthrough involves one or more evaluators exploring an interface, prototype, or paper mock-up by going through a pre-determined set of tasks and assessing the understandability and ease of learning for each task. During the walkthrough of a task, the evaluator(s) attempts to simulate a user's problem-solving process while examining each action required. The evaluator attempts to construct a credible success story for each step of the task. Otherwise, the evaluator constructs a detailed failure story.

Cognitive walkthroughs require intensive documentation effort. A modified version, cognitive jogthrough [Rowley and Rhoades 1992], was developed to expedite recording the walkthrough session. In cognitive jogthrough, the session is videotaped and logging software is used to mark key events. Thus, the videotape can be reviewed afterwards to document the session.

Pluralistic Walkthrough. This is a variation of the cognitive walkthrough inspection method wherein representative users, evaluators, and developers inspect the interface as a group. The goal of this method is to step through usage scenarios and discuss usability issues that arise in the scenario steps.

Heuristic Evaluation. In heuristic evaluation one or more evaluators independently evaluate an interface using a list of heuristics. The outcome of this evaluation is typically a list of possible usability problems. After the evaluators independently evaluate the interface, the evaluators aggregate their findings and associate severity ratings with each potential usability problem. Heuristic evaluation is the most informal inspection method [Nielsen and Mack 1994], mainly because it relies on a small set of usability criteria. It is also one of the main discount (i.e., cheap, fast, and easy to use) usability methods employed [Nielsen and Mack 1994].

Perspective-based Inspection. Perspective-based inspection [Zhang *et al.* 1998] is a variation of heuristic evaluation. For this method, evaluators divide a list of usability issues into different perspectives and focus on only one perspective or subset of heuristics during an inspection session. A perspective is a point of view consisting of a list of inspection questions and a specific procedure for conducting the inspection. Zhang *et al.* [1998] have shown that this

Method Class:	Inspection	
Automation Type:	Capture	
Method Type:	Cognitive Walkthrough - expert simulates user's problem solving (1 method)	
UE Method	**UI**	**Effort**
Software assists the expert with documenting a cognitive walkthrough	WIMP	F

Table 2.7: Synopsis of automated capture support for inspection methods.

approach improves the effectiveness of evaluators within each perspective as well as overall, in comparison to heuristic evaluation.

Feature Inspection. The purpose of this evaluation method is to inspect a feature set of a product and to analyze the availability, understandability, and other functionality aspects for each feature. Evaluators use a list of product features along with scenarios for such inspections. The documentation staff usually conducts feature inspections.

Formal Usability Inspection. Formal usability inspection is an adaptation of traditional software inspection to usability evaluation. The inspection procedure is fairly similar to heuristic evaluation and involves a diverse team of inspectors (e.g., developers, designers, documenters, trainers, technical support personnel, and possibly usability experts). The only difference is the formality associated with the inspection (i.e., assigned roles and a formal six-step process to follow).

Consistency Inspection. Evaluators use this method to determine a consistent interface appearance and functionality that they can then use to assess the consistency of interfaces across multiple products in a family.

Standards Inspection. In this inspection method an evaluator compares components of an interface to a list of industry standards to assess the interface's compliance with these standards. This inspection method is usually aimed at ensuring a product's market conformance.

2.6.2 Inspection Methods: Automated Capture

Table 2.7 summarizes capture support for inspection methods – namely, a system developed to assist an evaluator with a cognitive walkthrough. During a cognitive walkthrough, an evaluator attempts to simulate a user's problem-solving process while examining UI tasks. At each step of a task, the evaluator assesses whether a user would succeed or fail to complete the step. Hence, the evaluator produces extensive documentation during this analysis. There was an early attempt to "automate" cognitive walkthroughs by prompting evaluators with walkthrough questions and enabling evaluators to record their analyses in HyperCard. Unfortunately, evaluators found this approach too cumbersome and time-consuming to employ [Rieman *et al.* 1991].

2.6.3 Inspection Methods: Automated Analysis

Table 2.8 provides a synopsis of automated analysis methods for inspection-based usability evaluation, discussed in detail in the remainder of this section. All of the methods require minimal effort to employ; this is denoted with a blank entry in the effort column. Support available for WIMP and Web UIs is discussed separately.

Method Class:	Inspection	
Automation Type:	Analysis	
Method Type:	Guideline Review - expert checks guideline conformance (8 methods)	
UE Method	**UI**	**Effort**
Use quantitative screen measures for analysis (AIDE, [Parush *et al.* 1998])	WIMP	
Analyze terminology and consistency of UI elements (Sherlock)	WIMP	
Analyze the structure of Web pages (Rating Game, HyperAT, Gentler)	Web	
Use guidelines for analysis (WebSAT)	Web	
Analyze the scanning path of a Web page (Design Advisor)	Web	

Table 2.8: Synopsis of automated analysis support for inspection methods.

Inspection Methods: Automated Analysis – WIMP UIs

Several quantitative measures have been proposed for evaluating interfaces. Tullis [1983] derived size measures (Overall Density, Local Density, Number of Groups, Size of Groups, Number of Items, and Layout Complexity). [Streveler and Wasserman 1984] proposed "boxing," "hot-spot," and "alignment" analysis techniques. These early techniques were designed for alphanumeric displays, while more recent techniques evaluate WIMP interfaces. Vanderdonckt and Gillo [1994] proposed five visual techniques (Physical Composition, Association and Dissociation, Ordering, and Photographic Techniques), which identified more visual design properties than traditional balance, symmetry and alignment measures. Rauterberg [1996a] proposed and validated four measures (Functional Feedback, Interactive Directness, Application Flexibility, and Dialog Flexibility) to evaluate WIMP UIs. Quantitative measures have been incorporated into automated layout tools [Bodart *et al.* 1994; Kim and Foley 1993] as well as several automated analysis tools [Mahajan and Shneiderman 1997; Parush *et al.* 1998; Sears 1995], discussed immediately below.

Parush *et al.* [1998] developed and validated a tool for computing the complexity of dialog boxes implemented with Microsoft Visual Basic. The tool considers changes in the size of screen elements, the alignment and grouping of elements, as well as the utilization of screen space in its calculations. Usability studies demonstrated that tool results can be used to decrease screen search time and ultimately to improve screen layout. AIDE (semi-Automated Interface Designer and Evaluator) [Sears 1995] is a more advanced tool that helps designers assess and compare different design options using quantitative task-sensitive and task-independent metrics, including efficiency (i.e., distance of cursor movement), vertical and horizontal alignment of elements, horizontal and vertical balance, and designer-specified constraints (e.g., position of elements). AIDE also employs an optimization algorithm to automatically generate initial UI layouts. Studies with AIDE showed it to provide valuable support for analyzing the efficiency of a UI and incorporating task information into designs.

Sherlock [Mahajan and Shneiderman 1997] is another automated analysis tool for Windows interfaces. Rather than assessing ergonomic factors, it focuses on task-independent consistency checking (e.g., same widget placement and labels) within the UI or across multiple UIs; user studies have shown a 10–25% speedup for consistent interfaces [Mahajan and Shneiderman 1997]. Sherlock evaluates visual properties of dialog boxes, terminology (e.g., identify confusing terms and check spelling), as well as button sizes and labels. Sherlock evaluates any Windows UI that has been translated into a special canonical format file; this file contains GUI object descriptions. Currently,

there are translators for Microsoft Visual Basic and Microsoft Visual C++ resource files.

Inspection Methods: Automated Analysis – Web UIs

The Rating Game [Stein 1997] is an automated analysis tool that attempts to measure the quality of a set of Web pages using a set of easily measurable features. These include: an information feature (word to link ratio), a graphics feature (number of graphics on a page), a gadgets feature (number of applets, controls, and scripts on a page), and so on. The tool reports these raw measures without providing guidance for improving a Web page.

Two authoring tools from Middlesex University, HyperAT [Theng and Marsden 1998] and Gentler [Thimbleby 1997], perform a similar structural analysis at the site level. The goal of the Hypertext Authoring Tool (HyperAT) is to support the creation of well-structured hyperdocuments. It provides a structural analysis which focuses on verifying that the breadths and depths within a page and at the site level fall within thresholds (e.g., depth less than three levels). (HyperAT also supports inferential analysis of server log files similarly to other log file analysis techniques; see Section 2.5.3.) Gentler [Thimbleby 1997] provides similar structural analysis but focuses on maintenance of existing sites rather than design of new ones.

The Web Static Analyzer Tool (SAT) [Scholtz and Laskowski 1998], part of the NIST WebMetrics suite of tools, assesses static HTML according to a number of usability guidelines, such as whether all graphics contain ALT tags, the average number of words in link text, and the existence of at least one outgoing link on a page. Currently, WebSAT only processes individual pages and does not suggest improvements [Chak 2000]. Future plans for this tool include adding the ability to inspect the entire site more holistically to identify potential problems in interactions between pages.

Unlike other analysis approaches, the Design Advisor [Faraday 2000] enables visual analysis of Web pages. The tool uses empirical results from eye tracking studies designed to assess the attentional effects of various elements, such as animation, images, and highlighting, in multimedia presentations [Faraday and Sutcliffe 1998]; these studies found motion, size, images, color, text style, and position to be scanned in this order. The Design Advisor determines and superimposes a scanning path on a Web page where page elements are numbered to indicate the order in which elements will be scanned. It currently does not provide suggestions for improving scanning paths.

Inspection Methods: Automated Analysis – Discussion

Table 2.8 summarizes automated analysis methods discussed in this section. All of the WIMP approaches are highly effective at checking for guidelines that can be operationalized. These include computing quantitative measures (e.g., the size of screen elements, screen space usage, and efficiency) and checking consistency (e.g., same widget size and placement across screens). All of the tools have also been empirically validated. However, the tools cannot assess UI aspects that cannot be operationalized, such as whether the labels used on elements will be understood by users. For example, [Farenc *et al.* 1999] show that only 78% of a set of established ergonomic guidelines could be operationalized in the best case scenario and only 44% in the worst case. All methods also suffer from limited applicability (interfaces developed with Microsoft Visual Basic or Microsoft Visual C). The tools appear to be straight forward to learn and use, provided the UI is developed in the appropriate environment.

The Rating Game, HyperAT, and Gentler compute and report a number of statistics about a page (e.g., number of links, graphics, and words). However, the effectiveness of these structural analyses is questionable, since the thresholds have not been empirically validated. Although there

Method Class:	Inspection	
Automation Type:	Critique	
Method Type:	Guideline Review - expert checks guideline conformance (11 methods)	
UE Method	**UI**	**Effort**
Use guidelines for critiquing (KRI/AG, IDA, CHIMES, Ergoval)	WIMP	
Use guidelines for critiquing and modifying a UI (SYNOP)	WIMP	M
Check HTML syntax (Weblint, Dr. Watson)	Web	
Use guidelines for critiquing (Lift Online, Lift Onsite, Bobby, WebEval)	Web	

Table 2.9: Synopsis of automated critique support for inspection methods.

have been some investigations into breadth and depth tradeoffs for the Web [Larson and Czerwinski 1998; Zaphiris and Mtei 1997], general thresholds still remain to be established. Although WebSAT helps designers adhere to good coding practices, these practices have not been shown to improve usability. There may be some indirect support for these methods through research aimed at identifying aspects that affect Web site credibility [Fogg *et al.* 2001; Kim and Fogg 1999; Fogg *et al.* 2000], since credibility affects usability and vice versa. A survey of over 1,400 Web users as well as an empirical study indicated that typographical errors, ads, broken links, and other aspects impact credibility; some of these aspects can be detected by automated UE tools, such as WebSAT. All of these approaches are easy to use, learn, and apply to all Web UIs.

The visual analysis supported by the Design Advisor could help designers improve Web page scanning. It requires a special Web browser for use, but is easy to use, learn, and apply to basic Web pages (i.e., pages that don't use scripts, applets, Macromedia Flash, or other non-HTML technology). Heuristics employed by this tool were developed based on empirical results from eye tracking studies of multimedia presentations, but have not been empirically validated for Web pages.

2.6.4 Inspection Methods: Automated Critique

Critique systems give designers clear directions for conforming to violated guidelines and consequently improving usability. As mentioned above, following guidelines is difficult, especially when there is a large number of guidelines to consider. Automated critique approaches, especially ones that modify a UI [Balbo 1995], provide the highest level of support for adhering to guidelines.

Table 2.9 provides a synopsis of automated critique methods discussed in the remainder of this section. All but one method, SYNOP, require minimal effort to employ; this is denoted with a blank entry in the effort column. Support available for WIMP and Web UIs is discussed separately.

Inspection Methods: Automated Critique – WIMP UIs

The KRI/AG tool (Knowledge-based Review of user Interface) [Lowgren and Nordqvist 1992] is an automated critique system that checks the guideline conformance of X Window interface designs created using the TeleUSE UIMS [Lee 1997]. KRI/AG contains a knowledge base of guidelines and style guides, including the Smith and Mosier guidelines [Smith and Mosier 1986] and the Motif style guides [Open Software Foundation 1991]. It uses this information to automatically critique a UI design and generate comments about possible flaws in the design. IDA (user

Interface Design Assistance) [Reiterer 1994] also embeds rule-based (i.e., expert system) guideline checks within a UIMS. SYNOP [Balbo 1995] is a similar automated critique system that performs a rule-based critique of a control system application. SYNOP also modifies the UI model based on its evaluation. CHIMES (Computer-Human Interaction ModElS) [Jiang *et al.* 1993] assesses the degree to which NASA's space-related critical and high risk interfaces meet human factors standards.

Unlike KRI/AG, IDA, SYNOP, and CHIMES, Ergoval [Farenc and Palanque 1999] is widely applicable to WIMP UIs on Windows platforms. It organizes guidelines into an object-based framework (i.e., guidelines that are relevant to each graphical object) to bridge the gap between the developer's view of an interface and how guidelines are traditionally presented (i.e., checklists). This approach is being incorporated into a petri net environment [Palanque *et al.* 1999] to enable guideline checks throughout the development process.

Inspection Methods: Automated Critique – Web UIs

Several automated critique tools use guidelines for Web site usability checks. The World Wide Web Consortium's HTML Validation Service [World Wide Web Consortium 2000] checks that HTML code conforms to standards. Weblint [Bowers 1996] and Dr. Watson [Addy & Associates 2000] also check HTML syntax and in addition verify links. Dr. Watson also computes download speed and spell checks text.

UsableNet's LIFT Online and LIFT Onsite [Usable Net 2000] perform usability checks similarly to WebSAT (discussed in Section 2.6.3) as well as checking for use of standard and portable link, text, and background colors, the existence of stretched images, and other guideline violations. LIFT Online suggests improvements, while LIFT Onsite guides users through making suggested improvements. According to [Chak 2000], these two tools provide valuable guidance for improving Web sites. Bobby [Clark and Dardailler 1999; Cooper 1999] is another HTML analysis tool that checks Web pages for their accessibility [Web Accessibility Initiative 1999] to people with disabilities.

Conforming to the guidelines embedded in these tools can potentially eliminate usability problems that arise due to poor HTML syntax (e.g., missing page elements) or guideline violations. As previously discussed, research on Web site credibility [Fogg *et al.* 2001; Kim and Fogg 1999; Fogg *et al.* 2000] possibly suggests that some of the aspects assessed by these tools, such as broken links and other errors, may also affect usability due to the relationship between usability and credibility. However, [Ratner *et al.* 1996] question the validity of HTML usability guidelines, since most have not been subjected to a rigorous development process as established guidelines for WIMP interfaces. Analysis of 21 HTML guidelines showed little consistency among them, with 75% of recommendations appearing in only one style guide. Furthermore, only 20% of HTML-relevant recommendations from established WIMP guidelines existed in the 21 HTML style guides. WebEval [Scapin *et al.* 2000] is one automated critique approach being developed to address this issue. Similarly to Ergoval (discussed above), it provides a framework for applying established WIMP guidelines to relevant HTML components. Even with WebEval, some problems, such as whether text will be understood by users, are difficult to detect automatically.

Inspection Methods: Automated Critique – Discussion

Table 2.9 summarizes automated critique methods discussed in this section. All of the WIMP approaches are highly effective at suggesting UI improvements for those guidelines that can be operationalized. These include checking for the existence of labels for text fields, listing menu

options in alphabetical order, and setting default values for input fields. However, they cannot assess UI aspects that cannot be operationalized, such as whether the labels used on elements will be understood by users. As previously discussed, [Farenc *et al.* 1999] show that only 78% of a set of established ergonomic guidelines could be operationalized in the best case scenario and only 44% in the worst case. Another drawback of approaches that are not embedded within a UIMS (e.g., SYNOP) is that they require considerable modeling and learning effort on the part of the evaluator. All methods, except Ergoval, also suffer from limited applicability.

As previously discussed, conforming to the guidelines embedded in HTML analysis tools can potentially eliminate usability problems that arise due to poor HTML syntax (e.g., missing page elements) or guideline violations. However, [Ratner *et al.* 1996] question the validity of HTML usability guidelines, since most have not been subjected to a rigorous development process as established guidelines for WIMP interfaces and have little consistency among them. Brajnik [2000] surveyed eleven automated Web site analysis methods, including Bobby and Lift Online. The author's survey revealed that these tools address only a sparse set of usability features, such as download time, presence of alternative text for images, and validation of HTML and links. Other usability aspects, such as consistency and information organization are unaddressed by existing tools.

All of the Web critique tools are applicable to basic HTML pages and appear to be easy to use and learn. They also enable ongoing assessment, which can be extremely beneficial after making changes.

2.7 Inquiry Methods

Similarly to usability testing approaches, inquiry methods require feedback from users and are often employed during usability testing. However, the focus is not on studying specific tasks or measuring performance. Rather the goal of these methods is to gather subjective impressions (i.e., preferences or opinions) about various aspects of a UI. Evaluators also use inquiry methods, such as surveys, questionnaires, and interviews, to gather supplementary data after a system is released; this is useful for improving the interface for future releases. In addition, evaluators use inquiry methods for needs assessment early in the design process.

Inquiry methods vary based on whether the evaluator interacts with a user or a group of users or whether users report their experiences using questionnaires or usage logs, possibly in conjunction with screen snapshots. Automation has been used predominately to capture subjective impressions during formal or informal interface use.

2.7.1 Inquiry Methods: Non-automated

This section provides a synopsis of non-automated inquiry method types. The method type and method are the same in all cases. All of the methods require formal or informal interface use. In addition, all of the methods have been or could be applied to WIMP and Web UIs. Unless otherwise specified, most discussions are based on [Dix *et al.* 1998; Hom 1998; Human Factors Engineering 1999b; Nielsen 1993; Shneiderman 1998].

Contextual Inquiry. Contextual inquiry is a structured field interviewing method based on three core principles: 1. understanding the context in which a product is used is essential for its successful design; 2. the user is a partner in the design process; and 3. the usability design process must have a focus. Given these guiding principles, an evaluator attempts to discover

users' needs through on-site free-flow interviewing. A contextual inquiry is usually a long-term study possibly lasting a year or more. It is usually conducted in the early stages of system development.

Field Observation. Field observation is similar to, but less structured than contextual inquiry. Furthermore, a field observation is typically conducted for a released system. For this method, evaluators visit the representative users' workplace and observe them working with the system. This enables the evaluator to understand how users are using the system to accomplish their tasks as well as the mental model the users have of the system. During this visit, the evaluator may also interview users about their jobs and other aspects about the way they use the product. Furthermore, evaluators may collect artifacts. This method is also described as an ethnographic study.

Focus Groups. A focus group is a meeting of about six to nine users wherein users discuss issues relating to the system. The evaluator plays the role of the moderator (i.e., asks about pre-determined issues) and gathers the needed information from the discussion. This is valuable for improving the usability of future releases.

Interviews. An interview is essentially a discussion session between a single user and an interviewer. During an interview, an evaluator asks a user a series of questions about system issues to guide the discussion. The evaluator can use either an unstructured or structured interviewing method. In unstructured interviewing there is no well-defined agenda, and the objective is to obtain information on procedures adopted by the user and the user's expectations of the system. Structured interviewing has a specific, pre-determined agenda with specific questions to guide and direct the interview. Unstructured interviewing is more of a conversation, while structured interviewing is more of an interrogation.

Surveys. During a survey, an evaluator asks a user pre-determined questions and records responses. However, surveys are not as formal or structured as interviews.

Questionnaires. A questionnaire is a measurement tool designed to assess a user's subjective satisfaction with an interface. It is a list of questions that are distributed to users for responses. Responses on a questionnaire are usually quantitative (e.g., ratings on a 5-point scale). One example questionnaire is the Questionnaire for User Interaction Satisfaction (QUIS) [Harper and Norman 1993; Human Factors Engineering 1999b]. QUIS contains questions to rate 27 system attributes on a 10-point scale, including overall system satisfaction, screen visibility, terminology, system information, learning factors, and system capabilities.

Self-reporting Logs. Self-reporting logs is a paper-and-pencil form of logging wherein users write down their actions, observations, and comments on a system and then send them to the evaluator. This method is most appropriate in early stages of development.

Screen Snapshots. Screen snapshots are usually captured by a participant in conjunction with other journaling methods, such as self-reporting logs. Basically, participants take screen snapshots at various times during execution of pre-determined tasks.

User Feedback. User feedback is a means for users to give comments on the system as necessary or at their convenience. For some systems, it may be possible to make a feedback button or command accessible within the interface. It is also possible to allow users to submit feedback via electronic mail, bulletin boards, or Web sites.

Method Class:	Inquiry	
Automation Type:	Capture	
Method Type:	Questionnaires - user provides answers to specific questions (2 methods)	
UE Method	**UI**	**Effort**
Questionnaire embedded within the UI (UPM)	WIMP	IF
HTML forms-based questionnaires (e.g., WAMMI, QUIS, SUMI, or NetRaker)	WIMP, Web	IF

Table 2.10: Synopsis of automated capture support for inquiry methods.

2.7.2 Inquiry Methods: Automated Capture

Table 2.10 provides a synopsis of capture methods developed to assist users with completing questionnaires. Software tools enable the evaluator to collect subjective usability data and possibly make improvements throughout the life of an interface. Questionnaires can be embedded within a WIMP UI to facilitate the response capture process. Typically dialog boxes prompt users for subjective input and process responses (e.g., save data to a file or email data to the evaluator). For example, UPM (the User Partnering Module) [Abelow 1993] uses event-driven triggers (e.g., errors or specific command invocations) to ask users specific questions about their interface usage. This approach allows the evaluator to capture user reactions while they are still fresh.

The Web inherently facilitates the capture of questionnaire data using forms. Users are typically presented with an HTML page for entering data, and a program on the Web server (e.g., a CGI script) processes responses. Several validated questionnaires are available in Web format, including QUIS (Questionnaire for User Interaction Satisfaction) [Harper and Norman 1993] and SUMI (Software Usability Measurement Inventory) [Porteous *et al.* 1993] for WIMP interfaces and WAMMI (Website Analysis and MeasureMent Inventory) [Kirakowski and Claridge 1998] for Web interfaces. NetRaker's [NetRaker 2000] usability research tools enable evaluators to create custom HTML questionnaires and usability tests via a template interface and to view a graphical summary of results even while studies are in progress. NetRaker's tools include the NetRaker Index (a short usability questionnaire) for continuously gathering feedback from users about a Web site. Chak [2000] reports that NetRaker's tools are highly effective for gathering direct user feedback, but points out the need to address potential irritations caused by the NetRaker Index's pop-up survey window.

As previously discussed, automated capture methods represent an important first step toward informing UI improvements. Automation support for inquiry methods makes it possible to collect data quickly from a larger number of users than is typically possible without automation. However, these methods suffer from the same limitation of non-automated approaches – they may not clearly indicate usability problems due to the subjective nature of user responses. Furthermore, they do not support automated analysis or critique of interfaces. The real value of these techniques is that they are easy to use and widely applicable.

2.8 Analytical Modeling Methods

Analytical modeling complements traditional evaluation techniques like usability testing. Given some representation or model of the UI and/or the user, these methods enable the evaluator to inexpensively predict usability. A wide range of modeling techniques have been developed, and

they support different types of analyses. de Haan *et al.* [1992] classify modeling approaches into the following four categories:

- **Models for task environment analysis:** enable the evaluator to assess the mapping between the user's goals and UI tasks (i.e., how the user accomplishes these goals within the UI). ETIT (External Internal Task Mapping) [Moran 1983] is one example for evaluating the functionality, learnability, and consistency of the UI;
- **Models to analyze user knowledge:** enable the evaluator to use formal grammars to represent and assess knowledge required for interface use. AL (Action Language) [Reisner 1984] and TAG (Task-Action Grammar) [Payne and Green 1986] allow the evaluator to compare alternative designs and predict differences in learnability;
- **Models of user performance:** enable the evaluator to predict user behavior, mainly task completion time. GOMS analysis (Goals, Operators, Methods, and Selection rules) [John and Kieras 1996], CTA (Cognitive Task Analysis) [May and Barnard 1994], and (Programmmable User Models) PUM [Young *et al.* 1989] – support performance prediction; and
- **Models of the user interface:** enable the evaluator to represent the UI design at multiple levels of abstraction (e.g., syntactic and semantic levels) and assess this representation. CLG (Command Language Grammar) [Moran 1981] and ETAG (Extended Task-Action Grammar) [Tauber 1990] are two methods for representing and inspecting designs.

Models that focus on user performance, such as GOMS analysis, typically support quantitative analysis. The other approaches typically entail qualitative analysis and in some cases, such as TAG, support quantitative analysis as well. The survey only revealed automation support for methods that focus on user performance, including GOMS analysis, CTA, and PUM; this is most likely because performance prediction methods support quantitative analysis, which is easier to automate.

Automation has been predominately used to analyze task completion (e.g., execution and learning time) within WIMP UIs. Analytical modeling inherently supports automated analysis. The survey did not reveal analytical modeling techniques to support automated critique. Most analytical modeling and simulation approaches are based on the model human processor (MHP) proposed by Card *et al.* [1983]. GOMS analysis (Goals, Operators, Methods, and Selection Rules) is one of the most widely accepted analytical modeling methods based on the MHP [John and Kieras 1996]. Other methods based on the MHP employ simulation and will be discussed in Section 2.9.

2.8.1 Analytical Modeling Methods: Non-automated

This section provides a synopsis of non-automated modeling method types. Method types and methods are the same in all cases. All of the methods require model development and have only been employed for WIMP interfaces.

GOMS Analysis. The GOMS family of analytical modeling methods use a task structure consisting of Goals, Operators, Methods and Selection rules. Using this task structure along with validated time parameters for each operator, the methods enable predictions of task execution and learning times, typically for error-free expert performance. The four approaches in this family include the original GOMS method proposed by Card, Moran, and Newell (CMN-GOMS) [Card *et al.* 1983], the simpler keystroke-level model (KLM), the natural GOMS language (NGOMSL), and the critical path method (CPM-GOMS) [John and Kieras 1996].

These approaches differ in the task granularity modeled (e.g., keystrokes versus a high-level procedure), the support for alternative task completion methods, and the support for single goals versus multiple simultaneous goals.

Task-Environment Analysis. ETIT (External Internal Task Mapping) [Moran 1983] is an example method for studying the relationship between tasks completed in the user's domain and the mapping of these tasks into UI tasks. Terminology used for specific objects (e.g., character, word, or sentence) and operators on these objects (e.g., copy, split, or join) are first enumerated for each domain; then, mappings between the domains are determined. Establishing such a mapping enables the evaluator to make inferences about the functionality, learnability, and consistency of the UI. In addition, this method can be used for assessing the degree of knowledge transfer between alternative designs.

Knowledge Analysis. Knowledge analysis methods, such as AL (Action Language) [Reisner 1984] and TAG (Task-Action Grammar) [Payne and Green 1986], provide a formal grammar for representing and assessing knowledge required for converting specific user tasks into UI tasks. Both of these techniques assess usability by counting the number and depth of rules, but differ with respect to the formal grammar employed. AL uses a well-known notation for expressing computer science programming languages – Backus-Naur Form [Backus *et al.* 1964]. TAG uses a more sophisticated grammar, which produces more compact representations of rules. Measures computed from AL and TAG task representations can be used to compare alternative designs and to predict differences in learnability.

Design Analysis. Design analysis methods, such as CLG (Command Language Grammar) [Moran 1981] and ETAG (Extended Task-Action Grammar) [Tauber 1990], enable the evaluator to represent the UI design at multiple levels of abstraction (e.g., syntactic and semantic levels) and assess this representation. These methods are typically used for design specification prior to UI implementation. ETAG is a refinement of CLG that supports additional levels of analysis, such as specifying the syntax. ETAG has also been used for modeling user performance and knowledge.

2.8.2 Analytical Modeling Methods: Automated Analysis

Table 2.11 provides a synopsis of automated analysis methods discussed in the remainder of this section. The survey did not reveal analytical modeling methods for evaluating Web UIs.

Analytical Modeling Methods: Automated Analysis – WIMP UIs

Two of the major roadblocks to using GOMS have been the tedious task analysis and the need to calculate execution and learning times [Baumeister *et al.* 2000; Byrne *et al.* 1994; Hudson *et al.* 1999; Kieras *et al.* 1995]. These were originally specified and calculated manually or with generic tools such as spreadsheets. In some cases, evaluators implemented GOMS models in computational cognitive architectures, such as Soar or EPIC (discussed in Section 2.9). This approach actually added complexity and time to the analysis [Baumeister *et al.* 2000]. QGOMS (Quick and dirty GOMS) [Beard *et al.* 1996] and CATHCI (Cognitive Analysis Tool for Human Computer Interfaces) [Williams 1993] provide support for generating quantitative predictions, but still require the evaluator to construct GOMS models. Baumeister *et al.* [2000] studied these approaches and showed them to be inadequate for GOMS analysis.

Method Class:	Analytical Modeling	
Automation Type:	Analysis	
Method Type:	UIDE Analysis - conduct GOMS analysis within a UIDE (4 methods)	
UE Method	**UI**	**Effort**
Generate predictions for GOMS task models (QGOMS, CATHCI)	WIMP	M
Generate GOMS task models and predictions (USAGE, CRITIQUE)	WIMP	M
Method Type:	Cognitive Task Analysis - predict usability problems (1 method)	
UE Method	**UI**	**Effort**
Cognitive Task Analysis (CTA)	WIMP	M
Method Type:	Programmable User Models - write program that acts like a user (1 method)	
UE Method	**UI**	**Effort**
Programmable User Models (PUM)	WIMP	M

Table 2.11: Synopsis of automated analysis support for analytical modeling methods.

USAGE[5] (the UIDE System for semi-Automated GOMS Evaluation) [Byrne *et al.* 1994] and CRITIQUE (the Convenient, Rapid, Interactive Tool for Integrating Quick Usability Evaluations) [Hudson *et al.* 1999] provide support for automatically generating a GOMS task model and quantitative predictions for the model. Both of these tools accomplish this within a user interface development environment (UIDE). GLEAN (GOMS Language Evaluation and ANalysis) [Kieras *et al.* 1995] is another tool that generates quantitative predictions for a given GOMS task model (discussed in more detail in Section 2.9). These tools reduce the effort required to employ GOMS analysis and generate predictions that are consistent with models produced by experts. The major hindrance to wide application of these tools is that they operate on limited platforms (e.g., Sun machines), model low-level goals (e.g., at the keystroke level for CRITIQUE), do not support multiple task completion methods (even though GOMS was designed to support this), and rely on an idealized expert user model.

Cognitive Task Analysis (CTA) [May and Barnard 1994] uses a different modeling approach than GOMS analysis. GOMS analysis requires the evaluator to construct a model for each task to be analyzed. However, CTA requires the evaluator to input an interface description to an underlying theoretical model for analysis. The theoretical model, an expert system based on Interacting Cognitive Subsystems (ICS, discussed in Section 2.9), generates predictions about performance and usability problems similarly to a cognitive walkthrough. The system prompts the evaluator for interface details from which it generates predictions and a report detailing the theoretical basis of predictions. The authors refer to this form of analysis as "supportive evaluation."

The Programmable User Model (PUM) [Young *et al.* 1989] is an entirely different analytical modeling technique. In this approach, the designer is required to write a program that acts like a user using the interface; the designer must specify explicit sequences of operations for each task. Task sequences are then analyzed by an architecture (similar to the CTA expert system) that imposes approximations of psychological constraints, such as memory limitations. Constraint violations can be seen as potential usability problems. The designer can alter the interface design to resolve violations, and ideally improve the implemented UI as well. Once the designer successfully programs the architecture (i.e., creates a design that adheres to the psychological constraints),

[5]This is not to be confused with the UsAGE log file capture and analysis tool discussed in Section 2.5.

the model can then be used to generate quantitative performance predictions similarly to GOMS analysis. By making a designer aware of considerations and constraints affecting usability from the user's perspective, this approach provides clear insight into specific problems with a UI.

Analytical Modeling Methods: Automated Analysis – Discussion

Table 2.11 summarizes automated analysis methods discussed in this section. Analytical modeling approaches enable the evaluator to produce relatively inexpensive results to inform design choices. GOMS analysis has been shown to be applicable to all types of WIMP UIs and is effective at predicting usability problems. However, these predictions are limited to error-free expert performance in many cases although early accounts of GOMS considered error correction [Card *et al.* 1983]. The development of USAGE and CRITIQUE has reduced the learning time and effort required to apply GOMS analysis, but they suffer from limitations previously discussed. Tools based on GOMS may also require empirical studies to determine operator parameters in cases where these parameters have not been previously validated and documented.

Although CTA is an ideal solution for iterative design, it does not appear to be a fully-developed methodology. Two demonstration systems have been developed and effectively used by a group of practitioners as well as by a group of graduate students [May and Barnard 1994]. However, some users experienced difficulty with entering system descriptions, which can be a time consuming process. After the initial interface specification, subsequent analysis is easier because the demonstration systems store interface information. The approach appears to be applicable to all WIMP UIs. It may be possible to apply a more fully developed approach to Web UIs.

PUM is a programming approach, and thus requires considerable effort and learning time to employ. Although it appears that this technique is applicable to all WIMP UIs, its effectiveness is not discussed in detail in the literature.

Analytical modeling of Web UIs lags far behind efforts for WIMP interfaces. Many Web authoring tools, such as Microsoft FrontPage and Macromedia Dreamweaver, provide limited support for usability evaluation in the design phase (e.g., predict download time and check HTML syntax). This addresses only a small fraction of usability problems. While analytical modeling techniques are potentially beneficial, the survey did not uncover any approaches that address this gap in Web site evaluation. Approaches like GOMS analysis will not map as well to the Web domain, because it is difficult to predict how a user will accomplish the goals in a task hierarchy given that there are potentially many different ways to navigate a typical site. Another problem is GOMS' reliance on an expert user model (at least in the automated approaches), which does not fit the diverse user community of the Web. Hence, new analytical modeling approaches, such as a variation of CTA, are required to evaluate the usability of Web sites.

2.9 Simulation Methods

Simulation complements traditional UE methods and inherently supports automated analysis. Using models of the user and/or the interface design, computer programs simulate the user interacting with the interface and report the results of this interaction, in the form of performance measures and interface operations, for instance. Evaluators can run simulators with different parameters to study various UI design tradeoffs and thus make more informed decisions about UI implementation. Simulation is also used to automatically generate synthetic usage data for analysis with log file analysis techniques [Chi *et al.* 2000] or event playback in a UI [Kasik and George 1996]. Thus, simulation can also be viewed as supporting automated capture to some degree.

Method Class:	Simulation	
Automation Type:	Capture	
Method Type:	Genetic Algorithm Modeling - mimic novice user interaction (1 method)	
UE Method	**UI**	**Effort**
Genetic Algorithm Modeling ([Kasik and George 1996])	WIMP	
Method Type:	Information Scent Modeling - mimic Web site navigation (1 method)	
UE Method	**UI**	**Effort**
Information Scent Modeling ([Chi *et al.* 2000])	Web	M

Table 2.12: Synopsis of automated capture support for simulation methods.

2.9.1 Simulation Methods: Automated Capture

Table 2.12 provides a synopsis of the two automated capture methods discussed in this section. Kasik and George [1996] developed an automated technique for generating and capturing usage data; this data could then be used for driving tools that replay events (such as executing a log file) within Motif-based UIs. The goal of this work is to use a small number of input parameters to inexpensively generate a large number of usage traces (or test scripts) representing novice users. The evaluator can then use these traces to find weak spots, failures, and other usability problems.

To create novice usage traces, the designer initially produces a trace representing an expert using the UI; a scripting language is available to produce this trace. The designer can then insert deviation commands at different points within the expert trace. During trace execution, a genetic algorithm determines user behavior at deviation points, and in effect simulates a novice user learning by experimentation. Genetic algorithms consider past history in generating future random numbers; this enables the emulation of user learning. Altering key features of the genetic algorithm enables the designer to simulate other user models. Although currently not supported by this tool, traditional random number generation can also be employed to explore the outer limits of a UI, for example, by simulating completely random behavior.

Chi *et al.* [2000] developed a similar approach for generating and capturing navigation paths for Web UIs. This approach creates a model of an existing site that embeds information about the similarity of content among pages, server log data, and linking structure. The evaluator specifies starting points in the site and information needs (i.e., target pages) as input to the simulator. The simulator models a number of agents (i.e., hypothetical users) traversing the links and content of the site model. At each page, the model considers information "scent" (i.e., common keywords between an agent's goal and content on linked pages) in making navigation decisions. Navigation decisions are controlled probabilistically such that most agents traverse higher-scent links (i.e., closest match to information goal) and some agents traverse lower-scent links. Simulated agents stop when they reach the target pages or after an arbitrary amount of effort (e.g., maximum number of links or browsing time). The simulator records navigation paths and reports the proportion of agents that reached target pages.

The authors use these usage paths as input to the Dome Tree visualization methodology, an inferential log file analysis approach discussed in Section 2.5. The authors compared actual and simulated navigation paths for Xerox's corporate site and discovered a close match when scent is "clearly visible" (meaning links are not embedded in long text passages or obstructed by images). Since the site model does not consider actual page elements, the simulator cannot account for the impact of various page aspects, such as the amount of text or reading complexity, on navigation

Method Class:	Simulation	
Automation Type:	Analysis	
Method Type:	Petri Net Modeling - mimic user interaction from usage data (1 method)	
UE Method	**UI**	**Effort**
Petri Net Modeling (AMME)	WIMP	IF
Method Type:	Information Processor Modeling - mimic user interaction (9 methods)	
UE Method	**UI**	**Effort**
Employ a computational cognitive architecture for UI analysis (ACT-R, COGNET, EPIC, HOS, Soar, CCT, ICS, GLEAN)	WIMP	M
Employ a GOMS-like model to analyze navigation (Site Profile)	Web	M

Table 2.13: Synopsis of automated analysis support for simulation methods.

choices. Hence, this approach may enable only crude approximations of user behavior for sites with complex pages.

Simulation Methods: Automated Capture – Discussion

Table 2.12 summarizes automated capture methods discussed in this section. Without these techniques, the evaluator must anticipate all possible usage scenarios or rely on formal or informal interface use to generate usage traces. Formal and informal use limit UI coverage to a small number of tasks or to UI features that are employed in regular use. Automated techniques, such as the genetic algorithm approach, enable the evaluator to produce a larger number of usage scenarios and widen UI coverage with minimal effort.

The system developed by Kasik and George appears to be relatively straightforward to use, since it interacts directly with a running application and does not require modeling. Interaction with the running application also ensures that generated usage traces are plausible. Experiments demonstrated that it is possible to generate a large number of usage traces within an hour. However, an evaluator must manually analyze the execution of each trace to identify problems. The authors propose future work to automatically verify that a trace produced the correct result. The evaluator must also program an expert user trace, which could make the system difficult to use and learn. Currently, this tool is only applicable to Motif-based UIs.

The approach developed by Chi *et al.* is applicable to all Web UIs. It also appears to be straightforward to use and learn, since software produces the Web site model automatically. The evaluator must manually interpret simulation results; however, analysis could be facilitated with the Dome Tree visualization tool.

2.9.2 Simulation Methods: Automated Analysis

Table 2.13 provides a synopsis of the automated analysis methods discussed in the remainder of this section. Methods for WIMP and Web UIs are considered separately.

Simulation Methods: Automated Analysis – WIMP UIs

AMME [Rauterberg and Aeppili 1995] (see Section 2.5.2) is the only surveyed approach that constructs a WIMP simulation model (petri net) directly from usage data. Other methods are based on a model similar to the MHP and require the evaluator to conduct a task analysis (and

subsequently validate it with empirical data) to develop a simulator. Hence, AMME is more accurate, flexible (i.e., task and user independent), and simulates more detail (e.g., error performance and preferred task sequences). AMME simulates learning, user decisions, and task completion and outputs a measure of behavior complexity. Studies have shown that the behavior complexity measure correlates negatively with learning and interface complexity. Studies have also validated the accuracy of generated models with usage data [Rauterberg 1995]. AMME should be applicable to Web interfaces as well, since it constructs models from log files. Despite its advantages, AMME still requires formal interface use to generate log files for simulation studies.

The remaining WIMP simulation methods are based on sophisticated computational cognitive architectures – theoretical models of user behavior – similar to the MHP previously discussed. Unlike analytical modeling approaches, these methods attempt to approximate user behavior as accurately as possible. For example, the simulator may track the user's memory contents, interface state, and the user's hand movements during execution. This enables the simulator to report a detailed trace of the simulation run. Some simulation methods, such as CCT [Kieras and Polson 1985] (discussed below), can also generate predictions statically (i.e., without being executed) similarly to analytical modeling methods.

Pew and Mavor [Pew and Mavor 1998] provide a detailed discussion of computational cognitive architectures and an overview of many approaches, including five discussed below: ACT-R (Adaptive Control of Thought) [Anderson 1990; Anderson 1993], COGNET (COGnition as a NEtwork of Tasks) [Zachary *et al.* 1996], EPIC (Executive-Process Interactive Control) [Kieras *et al.* 1997], HOS (Human Operator Simulator) [Glenn *et al.* 1992], and Soar [Laird and Rosenbloom 1996; Polk and Rosenbloom 1994]. Here, CCT (Cognitive Complexity Theory) [Kieras and Polson 1985], ICS (Interacting Cognitive Subsystems) [Barnard 1987; Barnard and Teasdale 1991], and GLEAN (GOMS Language Evaluation and ANalysis) [Kieras *et al.* 1995] are also considered. Rather than describe each method individually, Table 2.14 summarizes the major characteristics of these simulation methods as discussed below.

Modeled Tasks. The surveyed models simulate the following three types of tasks: a user performing cognitive tasks (e.g., problem-solving and learning: COGNET, ACT-R, Soar, ICS); a user immersed in a human-machine system (e.g., an aircraft or tank: HOS); and a user interacting with a typical UI (EPIC, GLEAN, CCT).

Modeled Components. Some simulations focus solely on cognitive processing (ACT-R, COGNET) while others incorporate perceptual and motor processing as well (EPIC, ICS, HOS, Soar, GLEAN, CCT).

Component Processing. Task execution is modeled either as serial processing (ACT-R, GLEAN, CCT), parallel processing (EPIC, ICS, Soar), or semi-parallel processing (serial processing with rapid attention switching among the modeled components, giving the appearance of parallel processing: COGNET, HOS).

Model Representation. To represent the underlying user or system, simulation methods use either task hierarchies (as in a GOMS task structure: HOS, CCT), production rules (CCT, ACT-R, EPIC, Soar, ICS), or declarative/procedural programs (GLEAN, COGNET). CCT uses both a task hierarchy and production rules to represent the user and system models, respectively.

Predictions. The surveyed methods return a number of simulation results, including predictions of task performance (EPIC, CCT, COGNET, GLEAN, HOS, Soar, ACT-R), memory load

Parameter	UE Methods
Modeled Tasks	
problem-solving and/or learning	COGNET, ACT-R, Soar, ICS
human-machine system	HOS
UI interaction	EPIC, GLEAN, CCT
Modeled Components	
cognition	ACT-R, COGNET
perception, cognition & motor	EPIC, ICS, HOS, Soar, GLEAN, CCT
Component Processing	
serial	ACT-R, GLEAN, CCT
semi-parallel	COGNET, HOS
parallel	EPIC, ICS, Soar
Model Representation	
task hierarchy	HOS, CCT
production rules	CCT, ACT-R, EPIC, Soar, ICS
program	GLEAN, COGNET
Predictions	
task performance	EPIC, CCT, COGNET, GLEAN, HOS, Soar, ACT-R
memory load	ICS, CCT
learning	ACT-R, Soar, ICS, GLEAN, CCT
behavior	ACT-R, COGNET, EPIC

Table 2.14: Characteristics of WIMP simulation methods that are based on a variation of the MHP.

(ICS, CCT), learning (ACT-R, SOAR, ICS, GLEAN, CCT), or behavior predictions such as action traces (ACT-R, COGNET, EPIC).

These methods vary widely in their ability to illustrate usability problems. Their effectiveness is largely determined by the characteristics discussed (modeled tasks, modeled components, component processing, model representation, and predictions). Methods that are potentially the most effective at illustrating usability problems model UI interaction and all components (perception, cognition, and motor) processing in parallel, employ production rules, and report on task performance, memory load, learning, and simulated user behavior. Such methods would enable the most flexibility and closest approximation of actual user behavior. The use of production rules is important in this methodology, because it relaxes the requirement for an explicit task hierarchy, thus allowing for the modeling of more dynamic behavior, such as Web site navigation.

EPIC is the only simulation analysis method that embodies most of these ideal characteristics. It uses production rules and models UI interaction and all components (perception, cognition, and motor) processing in parallel. It reports task performance and simulated user behavior, but does not report memory load and learning estimates. Studies with EPIC have demonstrated that predictions for telephone operator and menu searching tasks closely match observed data. EPIC and all of the other methods require considerable learning time and effort to use. They are also applicable to a wide range of WIMP UIs.

Simulation Methods: Automated Analysis – Web UIs

The survey revealed only one simulation approach for analysis of Web interfaces – WebCriteria's Site Profile [Web Criteria 1999]. Unlike the other simulation approaches, it requires an

implemented interface for evaluation. Site Profile performs analysis in four phases: gather, model, analyze, and report. During the gather phase, a spider traverses a site (200-600 unique pages) to collect Web site data. This data is then used to construct a nodes-and-links model of the site. For the analysis phase, it uses an idealistic Web user model (called Max [Lynch *et al.* 1999]) to simulate a user's information seeking behavior; this model is based on prior research with GOMS analysis. Given a starting point in the site, a path, and a target, Max "follows" the path from the starting point to the target and logs measurement data. These measurements are used to compute an accessibility metric, which is then used to generate a report. This approach can be used to compare Web sites, provided that an appropriate navigation path is supplied for each.

The usefulness of this approach is questionable, since currently it only computes accessibility (navigation time) for the shortest path between specified start and destination pages using a single user model. Other measurements, such as freshness and page composition, also have questionable value in improving the Web site. [Brajnik 2000] showed Site Profile to support only a small fraction of the analysis supported by guideline review methods, such as WebSAT and Bobby (discussed in Section 2.6). [Chak 2000] also cautions that the accessibility measure should be used as an initial benchmark, not a highly-accurate approximation. Site Profile does not entail any learning time or effort on the part of the evaluator, since WebCriteria performs the analysis. The method is applicable to all Web UIs.

Simulation Methods: Automated Analysis – Discussion

Table 2.13 summarizes automated analysis methods discussed in this section. Unlike most evaluation approaches, simulation can be used prior to UI implementation in most cases (although AMME and WebCriteria's Site Profile are exceptions to this). Hence, simulation enables alternative designs to be compared and optimized before implementation.

It is difficult to assess the effectiveness of simulation methods, although there have been reports that show EPIC [Kieras *et al.* 1997] and GLEAN [Baumeister *et al.* 2000] to be effective. AMME appears to be the most effective method, since it is based on actual usage. AMME also enables ongoing assessment and could be widely used for WIMP and Web interfaces, provided log files and system models are available. EPIC is the only method based on the MHP that embodies the ideal simulator characteristics previously discussed. GLEAN is actually based on EPIC, so it has similar properties.

In general, simulation methods are more difficult to use and learn than other evaluation methods, because they require constructing or manipulating complex models as well as understanding the theory behind a simulation approach. Approaches based on the MHP are widely applicable to all WIMP UIs. Approaches that use production rules, such as EPIC, CCT, and Soar, could possibly be applied to Web UIs where task sequences are not as clearly defined as WIMP UIs. Soar has actually been adapted to model browsing tasks similar to Web browsing [Peck and John 1992].

2.10 Expanding Existing Approaches to Automating Usability Evaluation Methods

Automated usability evaluation methods have many potential benefits, including reducing the costs of non-automated methods, aiding in comparisons between alternative designs, and improving consistency in evaluation results. Numerous methods that support automation have been studied. Based on the methods surveyed, research to further develop log file analysis, guideline review, analytical modeling, and simulation techniques could result in several promising automated

Method Class: Testing		
Automation Type: Analysis		
Method Type: Log File Analysis - analyze usage data (20 methods)		
UE Method	**UI**	**Effort**
Use metrics during log file analysis (DRUM, MIKE UIMS, AMME)	WIMP	IF
Use metrics during log file analysis (Service Metrics, [Bacheldor 1999])	Web	IF
Use pattern matching during log file analysis (MRP)	WIMP	IF
Use task models during log file analysis (IBOT, QUIP, WebQuilt, KALDI, UsAGE)	WIMP	IF
Use task models and pattern matching during log file analysis (ÉMA, USINE, RemUSINE)	WIMP	IFM
Visualization of log files ([Guzdial *et al.* 1994])	WIMP	IF
Statistical analysis or visualization of log files (traffic- and time-based analyses, VISVIP, Starfield and Dome Tree visualizations)	Web	IF

Table 2.15: Synopsis of automated analysis support for usability testing methods. This is a repetition of Table 2.6.

techniques as discussed in more detail below. Chapter 3 discusses other promising approaches based on performance evaluation of computer systems.

2.10.1 Expanding Log File Analysis Approaches

The survey showed log file analysis to be a viable methodology for automated analysis of usage data. Table 2.15 summarizes current approaches to log file analysis. These approaches could be expanded and improved in the following three ways:

- Generating synthetic usage data for analysis;
- Using log files for comparing (i.e., benchmarking) comparable UIs; and
- Augmenting task-based pattern-matching approaches with guidelines to support automated critique.

Generating synthetic usage data for analysis. The main limitation of log file analysis is that it still requires formal or informal interface use to employ. One way to expand the use and benefits of this methodology is to leverage a small amount of test data to generate a larger set of plausible usage data. This is even more important for Web interfaces, since server logs do not capture a complete record of user interactions. The discussion included two simulation approaches, one using a genetic algorithm [Kasik and George 1996] and the other using information scent modeling [Chi *et al.* 2000] (see Section 2.9.1), that automatically generate plausible usage data. The genetic algorithm approach determines user behavior during deviation points in an expert user script, while the information scent model selects navigation paths by considering word overlap between links and web pages. Both of these approaches generate plausible usage traces without formal or informal interface use. These techniques also provide valuable insight on how to leverage real usage data from usability tests or informal use. For example, real data could also serve as input scripts for genetic algorithms; the evaluator could add deviation points to these.

Method Class:	Inspection	
Automation Type:	Analysis	
Method Type:	Guideline Review - expert checks guideline conformance (8 methods)	
UE Method	**UI**	**Effort**
Use quantitative screen measures for analysis (AIDE, [Parush *et al.* 1998])	WIMP	
Analyze terminology and consistency of UI elements (Sherlock)	WIMP	
Analyze the structure of Web pages (Rating Game, HyperAT, Gentler)	Web	
Use guidelines for analysis (WebSAT)	Web	
Analyze the scanning path of a Web page (Design Advisor)	Web	

Table 2.16: Synopsis of automated analysis support for inspection methods. This is a repetition of Table 2.8.

Using log files for comparing UIs. Real and simulated usage data could also be used to evaluate comparable WIMP UIs, such as word processors and image editors. Task sequences could comprise a usability benchmark (i.e., a program for measuring UI performance); this is similar to GOMS analysis of comparable task models. After mapping task sequences into specific UI operations in each interface, the benchmark could be executed within each UI to collect measurements. Representing this benchmark as a log file of some form would enable the log file to be executed within a UI by replay tools, such as: QC/Replay [Centerline 1999] for X Windows; UsAGE [Uehling and Wolf 1995] for replaying events within a UIMS (discussed in Section 2.5); or WinRunner [Mercury Interactive 2000] for a wide range of applications (e.g., Java and Oracle applications). This is a promising open area of research for evaluating comparable WIMP UIs. Chapter 3 explores this concept in more detail.

Augmenting task-based pattern-matching approaches with guidelines to support automated critique. Given a wider sampling of usage data, using task models and pattern matching during log file analysis is a promising research area to pursue. Task-based approaches that follow the USINE model in particular (i.e., compare a task model expressed in terms of temporal relationships to usage traces) provide the most support, among the methods surveyed. USINE outputs information to help the evaluator understand user behavior, preferences, and errors. Although the authors claim that this approach works well for WIMP UIs, it needs to be adapted to work for Web UIs where tasks may not be clearly-defined. Additionally, since USINE already reports substantial analysis data, this data could be compared to usability guidelines to support automated critique.

2.10.2 Expanding Guideline Review Approaches

Several guideline review methods for analysis of WIMP interfaces (see Table 2.16) could be augmented with guidelines to support automated critique. For example, AIDE (discussed in Section 2.6) provides the most support for evaluating UI designs. It computes a number of quantitative measures and also generates initial interface layouts. Guidelines, such as thresholds for quantitative measures, could also be incorporated into AIDE analysis to support automated critique.

Although there are several guideline review methods for analyzing and critiquing Web UIs (see Tables 2.16 and 2.17), existing approaches only cover a small fraction of usability aspects [Brajnik 2000] and have not been empirically validated. This dissertation presents an approach for developing Web design guidelines directly from empirical data.

Method Class:	Inspection	
Automation Type:	Critique	
Method Type:	Guideline Review - expert checks guideline conformance (11 methods)	
UE Method	**UI**	**Effort**
Use guidelines for critiquing (KRI/AG, IDA, CHIMES, Ergoval)	WIMP	
Use guidelines for critiquing and modifying a UI (SYNOP)	WIMP	M
Check HTML syntax (Weblint, Dr. Watson)	Web	
Use guidelines for critiquing (Lift Online, Lift Onsite, Bobby, WebEval)	Web	

Table 2.17: Synopsis of automated critique support for inspection methods. This is a repetition of Table 2.9.

2.10.3 Expanding Analytical Modeling Approaches

The survey showed that evaluation within a user interface development environment (UIDE) is a promising approach for automated analysis via analytical modeling. Table 2.18 summarizes current approaches to analytical modeling. UIDE analysis methods, such as CRITIQUE and GLEAN, could be augmented with guidelines to support automated critique. Guidelines, such as thresholds for learning or executing certain types of tasks, could assist the designer with interpreting prediction results and improving UI designs. Evaluation within a UIDE should also make it possible to automatically optimize UI designs based on guidelines.

Although UIDE analysis is promising, it is not widely used in practice. This may be due to the fact that most tools are research systems and have not been incorporated into popular commercial tools. This is unfortunate since incorporating analytical modeling and possibly simulation methods within a UIDE should mitigate some barriers to their use, such as being too complex and time consuming to employ [Bellotti 1988]. Applying such analysis approaches outside of these user interface development environments is an open research problem.

Cognitive Task Analysis provides some insight for analyzing UIs outside of a UIDE. Furthermore, CTA is a promising approach for automated analysis, provided more effort is spent to fully develop this methodology. This approach is consistent with analytical modeling techniques employed outside of HCI, such as in the performance evaluation of computer systems [Jain 1991] (see Chapter 3); this is because with CTA the evaluator provides UI parameters to an underlying model for analysis versus developing a new model to assess each UI. However, one of the drawbacks of CTA is the need to describe the interface to the system. Integrating this approach into a UIDE or UIMS should make this approach more tenable.

As previously discussed, analytical modeling approaches for Web UIs still remain to be developed. It may not be possible to develop new approaches using a paradigm that requires explicit task hierarchies. However, a variation of CTA may be appropriate for Web UIs.

2.10.4 Expanding Simulation Approaches

Table 2.19 summarizes current approaches to simulation analysis. The survey showed that existing simulations based on a human information processor model have widely different uses (e.g., modeling a user interacting with a UI or solving a problem). Thus, it is difficult to draw concrete conclusions about the effectiveness of these approaches. Simulation in general is a promising research area to pursue for automated analysis, especially for evaluating alternative designs.

Method Class:	Analytical Modeling	
Automation Type:	Analysis	
Method Type:	UIDE Analysis - conduct GOMS analysis within a UIDE (4 methods)	
UE Method	**UI**	**Effort**
Generate predictions for GOMS task models (QGOMS, CATHCI)	WIMP	M
Generate GOMS task models and predictions (USAGE, CRITIQUE)	WIMP	M
Method Type:	Cognitive Task Analysis - predict usability problems (1 method)	
UE Method	**UI**	**Effort**
Cognitive Task Analysis (CTA)	WIMP	M
Method Type:	Programmable User Models - write program that acts like a user user (1 method)	
UE Method	**UI**	**Effort**
Programmable User Models (PUM)	WIMP	M

Table 2.18: Synopsis of automated analysis support for analytical modeling methods. This is a repetition of Table 2.11.

Method Class:	Simulation	
Automation Type:	Analysis	
Method Type:	Petri Net Modeling - mimic user interaction from usage data (1 method)	
UE Method	**UI**	**Effort**
Petri Net Modeling (AMME)	WIMP	IF
Method Type:	Information Processor Modeling - mimic user interaction (9 methods)	
UE Method	**UI**	**Effort**
Employ a computational cognitive architecture for UI analysis (ACT-R, COGNET, EPIC, HOS, Soar, CCT, ICS, GLEAN)	WIMP	M
Employ a GOMS-like model to analyze navigation (Site Profile)	Web	M

Table 2.19: Synopsis of automated analysis support for simulation methods. This is a repetition of Table 2.13.

It is possible to use several simulation techniques employed in the performance analysis of computer systems, in particular trace-driven discrete-event simulation and Monte Carlo simulation [Jain 1991], to enable designers to perform what-if analyses with UIs (see Chapter 3). Trace-driven discrete-event simulations use real usage data to model a system as it evolves over time. Analysts use this approach to simulate many aspects of computer systems, such as the processing subsystem, operating system, and various resource scheduling algorithms. In the user interface field, all surveyed approaches use discrete-event simulation. However, AMME constructs simulation models directly from logged usage, which is a form of trace-driven discrete-event simulation. Similarly, other simulators could be altered to process log files as input instead of explicit task or user models, potentially producing more realistic and accurate simulations.

Monte Carlo simulations enable an evaluator to model a system probabilistically (i.e., sampling from a probability distribution is used to determine what event occurs next). Monte Carlo simulation could contribute substantially to automated analysis by eliminating the need for explicit task hierarchies or user models. Most simulations in this domain rely on a single user model, typically an expert user. Monte Carlo simulation would enable designers to perform what-if analysis and study design alternatives with many user models. The approach employed by [Chi *et al.* 2000] to simulate Web site navigation is a close approximation to Monte Carlo simulation.

2.11 Summary

This chapter provided an overview of usability evaluation and presented a taxonomy for comparing various methods. It also presented an extensive survey of the use of automation in WIMP and Web interface evaluation, finding that automation is used in only 36% of methods surveyed. Of all of the surveyed methods, only 29% are free from requirements of formal or informal interface use. All approaches that do not require formal or informal use, with the exception of guideline review, are based on analytical modeling or simulation.

It is important to keep in mind that automation of usability evaluation does not capture important qualitative and subjective information (such as user preferences and misconceptions) that can only be unveiled via usability testing, heuristic evaluation, and other standard inquiry methods. Nevertheless, simulation and analytical modeling should be useful for helping designers choose among design alternatives before committing to expensive development costs.

Furthermore, evaluators could use automation in tandem with what are usually non-automated methods, such as heuristic evaluation and usability testing. For example, an evaluator doing a heuristic evaluation could observe automatically-generated usage traces executing within a UI.

Adding automation to usability evaluation has many potential benefits, including reducing the costs of non-automated methods, aiding in comparisons between alternative designs, and improving consistency in usability evaluation. Research to further develop analytical modeling, simulation, guideline review, and log file analysis techniques could result in several promising automated techniques. The next chapter discusses new approaches that could be developed based on performance evaluation of computer systems.