The State of the Art in Automating Usability Evaluation of User Interfaces

MELODY Y. IVORY AND MARTI A. HEARST

University of California, Berkeley

Usability evaluation is an increasingly important part of the user interface design process. However, usability evaluation can be expensive in terms of time and human resources, and automation is therefore a promising way to augment existing approaches. This article presents an extensive survey of usability evaluation methods, organized according to a new taxonomy that emphasizes the role of automation. The survey analyzes existing techniques, identifies which aspects of usability evaluation automation are likely to be of use in future research, and suggests new ways to expand existing approaches to better support usability evaluation.

Categories and Subject Descriptors: H.1.2 [**Information Systems**]: User/Machine Systems—*human factors*; *human information processing*; H.5.2 [**Information Systems**]: User Interfaces—*benchmarking*; *evaluation/methodology*; *graphical user interfaces (GUI)*

General Terms: Human Factors

Additional Key Words and Phrases: Graphical user interfaces, taxonomy, usability evaluation automation, web interfaces

1. INTRODUCTION

Usability is the extent to which a computer system enables users, in a given context of use, to achieve specified goals effectively and efficiently while promoting feelings of satisfaction.[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].

[1] Adapted from ISO9241 [International Standards Organization 1999].

Usability evaluation is an important part of the overall user interface design process, which 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 usability problems in the interface; and

This research was sponsored in part by the Lucent Technologies Cooperative Research Fellowship Program, a GAANN fellowship, and Kaiser Permanente.

Authors' addresses: M. Y. Ivory, Computer Science Division, University of California, Berkeley, Berkeley, CA 94720-1776; email: ivory@CS.Berkeley.edu; M. A. Hearst, School of Information Management and Systems, University of California, Berkeley, Berkeley, CA 94720-4600; email: hearst@SIMS.Berkeley.edu.

—*Critique*: suggesting solutions or improvements to mitigate problems.

A wide range of usability evaluation techniques have been proposed, and a subset of these is currently in common use. Some evaluation techniques, such as formal user 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 [Jeffries et al. 1991; Molich et al. 1998, 1999; Nielsen 1993]. Two studies in particular, the first and second comparative user 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 eight independent usability testing teams for evaluations of two user interfaces. This result implies a lack of systematicity or predictability in the findings 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.

Automation of usability evaluation has several potential advantages over nonautomated evaluation, such as the following.

—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.

—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.

—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 nonautomated 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.

—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.

—Incorporating evaluation within the design phase of UI development, as opposed to being applied after implementation. This is important because evaluation with most nonautomated 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 we consider automation 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.

Despite the potential advantages, the space of usability evaluation automation is quite underexplored. In this article, we discuss the state of the art in usability evaluation automation, and highlight the approaches that merit further investigation. Section 2 presents a taxonomy for classifying UE automation, and Section 3 summarizes the application of this taxonomy to 132 usability methods. Sections 4 through 8 describe these methods in more detail, including our summative assessments of automation techniques. The results of this survey suggest promising ways to expand existing approaches to better support usability evaluation.

2. TAXONOMY OF USABILITY EVALUATION AUTOMATION

In this discussion, we make a distinction between WIMP (windows, icons, pointer, and mouse) interfaces and Web interfaces, in part because the nature of these interfaces differs and in part because the usability 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; we highlight their differences when relevant to usability evaluation.

Several surveys of UE methods for WIMP interfaces exist; Hom [1998] and Human Factors Engineering [1999] 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 reflects the automated aspects of UE methods.

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

—*Nonautomatic*: 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, keyboard and mouse actions";
—*Automatic Analysis*: methods that are "able to identify usability problems automatically"; and
—*Automatic Critic*: methods that "not only point out difficulties but propose improvements."

Balbo uses these categories to classify 13 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 nonautomated requirements of the UE methods.

2.1. Proposed Taxonomy

To facilitate our discussion of the state of automation in usability evaluation, we have grouped UE methods 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).

2.1.1. Method Class. We classify UE methods into five method classes as follows.

—*Testing*: an evaluator observes users 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*: users provide feedback on an interface via interviews, surveys, and the like.

—*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, whereas the latter two, analytical modeling and simulation, are quite similar to performance evaluation techniques used to analyze the performance of computer systems [Ivory 2001; Jain 1991].

2.1.2. Method Type. There is a wide range of evaluation methods within the testing, inspection, inquiry, analytical modeling, and simulation classes. Rather than discuss each method individually, we group related methods into method types; this type typically describes how evaluation is performed. We present method types in Sections 4 through 8.

2.1.3. Automation Type. We adapted Balbo's automation taxonomy (described above) to specify which aspect of a usability evaluation method is automated.

—*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; and

—*Critique*: software automates analysis and suggests improvements.

2.1.4. Effort Level. We also expanded Balbo's automation taxonomy to include consideration of a method's nonautomated requirements. We augment each UE method with an attribute called *effort level*; this indicates the human effort required for method execution.

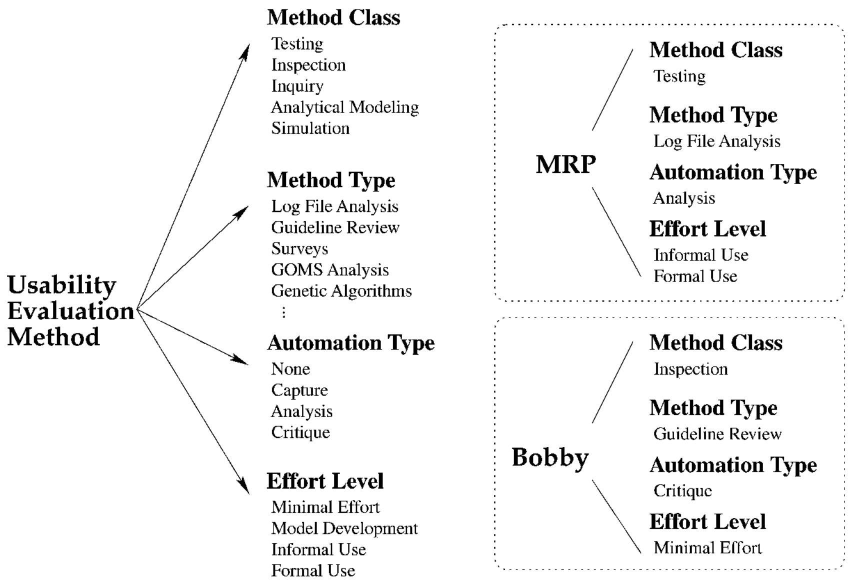


Fig. 1. Summary of our taxonomy for classifying usability evaluation methods. The right side of the figure demonstrates the taxonomy with two evaluation methods discussed in later sections.

—*Minimal Effort*: does not require interface usage or modeling.

—*Model Development*: requires the evaluator to develop a UI model and/or a user model in order to employ the method.

—*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 employed.

2.1.5. Summary. Figure 1 provides a synopsis of our 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, and surveys); an automation type (none, capture, analysis, and critique); and an effort level (minimal, model, informal, and formal). In the remainder of this article, we use this taxonomy to analyze evaluation methods.

3. OVERVIEW OF USABILITY EVALUATION METHODS

We surveyed 75 UE methods applied to WIMP interfaces, and 57 methods applied to Web UIs. Of these 132 methods, only 29 apply to both Web and WIMP UIs. We determined the applicability of each method based on the types of interfaces a method was used to evaluate in the literature and our judgment of whether the method could be used with other types of interfaces. Table I combines 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 2 through 5 depicts specific UE methods along with the automation support available and the effort required to employ automation. For some UE methods, we discuss more than one approach;

Table I. Automation Support for WIMP and Web UE Methods[a]

Method Class	Automation Type			
Method 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)			
Codiscovery Learning	F (1)			
Performance Measurement	F (1)	F (7)		
Log File Analysis			IFM (19)*	
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)			
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)		
Automation Type				
Total	30	6	8	1
Percent	67%	13%	18%	2%

[a]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).
*Indicates that either formal or informal interface use is required. In addition, a model may be used in the analysis.
†Indicates that methods may or may not employ a model.

hence, we show the number of methods surveyed in parentheses beside the effort level. Some approaches provide automation support for multiple method types (see Appendix A). Table I contains 110 methods because some methods are applicable to multiple method types; we also only depict methods applicable to both WIMP and Web UIs once. Table II 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

Table II. Descriptions of the WIMP and Web UE Method Types Depicted in Table I

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
Codiscovery Learning	two users collaborate
Performance Measurement	tester 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 colocated 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 violations of heuristics
Perspective-Based Inspection	expert conducts narrowly focused heuristic evaluation
Feature Inspection	expert evaluates product features
Formal Usability Inspection	expert conducts 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

in the Web domain than for WIMP interfaces (2 vs. 16 methods). Appendix A shows the information in Table I separated by UI type.

Table I shows that automation in general is greatly underexplored. Methods without automation support represent 67% of the methods surveyed, and methods with automation support collectively represent only 33%. Of this 33%, capture methods represent 13%, analysis methods represent 18%, 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 employ simulation to generate usage data for subsequent analysis. Overall, only 29% of all of the methods surveyed (nonautomated and automated) do not require formal or informal interface use to employ.

To provide the fullest automation support, software would have to critique interfaces without requiring formal or informal use. Our survey found that this level of automation has been developed for only one method type: guideline review (e.g., Farenc and Palanque [1999], Lowgren and Nordvist [1992], and Scholtz and Laskowski [1998]). Guideline review methods automatically detect and report usability violations and then make suggestions for fixing them (discussed further in Section 5).

Of those methods that support the next level of automation—analysis—Table I shows 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, we make distinctions where necessary about a method's applicability. Our discussion also includes our assessments of automated capture, analysis, and critique techniques using the criteria:

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

We highlight the effectiveness, ease of use, ease of learning, and applicability of automated methods in our discussion of each method class. Ivory [2001] provides a detailed discussion of all nonautomated and automated evaluation methods surveyed.

4. AUTOMATING USABILITY TESTING METHODS

Usability testing with real participants is a fundamental usability 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 predetermined set of tasks while the tester records the results of the participants' work. The tester then uses these results to determine how well the interface supports users' task completion as well as other measures, such as 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 these data according to some metrics or a model (referred to as log file analysis in Table I). In rare cases methods support both automated capture and analysis of usage data [Al-Qaimari and McRostie 1999; Uehling and Wolf 1995].

4.1. Automating Usability Testing Methods: Capture Support

Many usability testing methods require the recording of the 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 (e.g., keystrokes) and information that is meaningful but difficult to automatically 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 is a software library that provides high-level abstractions for specifying portable and consistent interface models that are then compiled into UI implementations [Olsen, Jr. 1992]. Table III provides a synopsis of automated capture methods discussed in the remainder of this section. We discuss support available for WIMP and Web UIs separately.

Table III. Synopsis of Automated Capture Support for Usability Testing Methods[a]

Method Class: Usability Testing		
Automation Type: Capture		
Method Type: Performance Measurement—record usage data during test (7 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
Method Type: Remote Testing—tester and user are not colocated (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

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

4.1.1. Automating Usability Testing Methods: Capture Support—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)[2] [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) be available during playback as were 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 does not require special equipment. Both KALDI and UsAGE also support log file analysis (see Section 4.2).

Remote testing methods enable testing between a tester and participant who are not colocated. In this case the evaluator is not able to observe the participant directly, but can gather data about the process over a computer network. Remote testing methods are distinguished according to whether a tester observes the participant during testing. 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 such as the question-asking or thinking-aloud protocols that require such interaction.

The tester does not observe the participant 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

[2] This method is not to be confused with the UsAGE analytical modeling approach discussed in Section 7.

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 in order to record user 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.

4.1.2. Automating Usability Testing Methods: Capture Support—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., from where the user came). 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 4.2.) Using these 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 noninstrumented 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 timestamp, 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.

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 nonautomated 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.

4.1.3. Automating Usability Testing Methods: Capture Support—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, discussed below.

Table III 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 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, whereas 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. 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.

4.2. Automating Usability Testing Methods: Analysis Support

Log file analysis methods automate analysis of data captured during formal or informal interface use. Since Web servers

Table IV. Synopsis of Automated Analysis Support for Usability Testing Methods[a]

Method Class: Usability Testing		
Automation Type: Analysis		
Method Type: Log File Analysis—analyze usage data (19 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, 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

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

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]. Our survey reveals four general approaches for analyzing WIMP and Web log files: metric-based, pattern-matching, task-based, and inferential. Table IV provides a synopsis of automated analysis methods discussed in the remainder of this section. We discuss support available for the four general approaches separately.

4.2.1. Automating Usability Testing Methods: Analysis Support—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, 1995b; Rauterberg and Aeppili 1995]. DRUM enables the evaluator to review a videotape 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 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) in order 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 versus 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 8) 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 [1999] and others [Bacheldor 1999] allow evaluators to pinpoint performance bottlenecks, such as slow server response time, that may have a negative impact on 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.

4.2.2. Automating Usability Testing Methods: Analysis Support—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. We discuss these methods immediately below.

4.2.3. Automating Usability Testing Methods: Analysis Support—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-based 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- and trace-based information into directed graphs (see Figure 2). 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. Currently, the evaluator must program the UI to collect the necessary usage data, and must manually analyze the graphs to identify usability problems.

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 play back synchronized screen shots. It depicts logs graphically in order to facilitate analysis.

UsAGE [Uehling and Wolf 1995], which also supports logging usage data within a

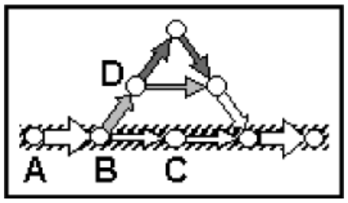


Fig. 2. QUIP usage profile contrasting task flows for two users to the designer's task flow (diagonal shading) [Helfrich and Landay 1999]. Each node represents a user action, and arrows indicate actions taken by users. The width of arrows denotes the fraction of users completing actions, and the color of arrows reflects the average time between actions (darker colors correspond to longer times). Reprinted with permission of the authors.

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 dataflow 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.

4.2.4. Automating Usability Testing Methods: Analysis Support—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., clickstreams and page-view durations) [Drott 1998; Fuller and de Graaff 1996; Sullivan 1997; Theng and Marsden 1998]. Some methods require manual preprocessing or filtering of the logs before analysis. Furthermore, the evaluator must interpret reported measures in order 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].

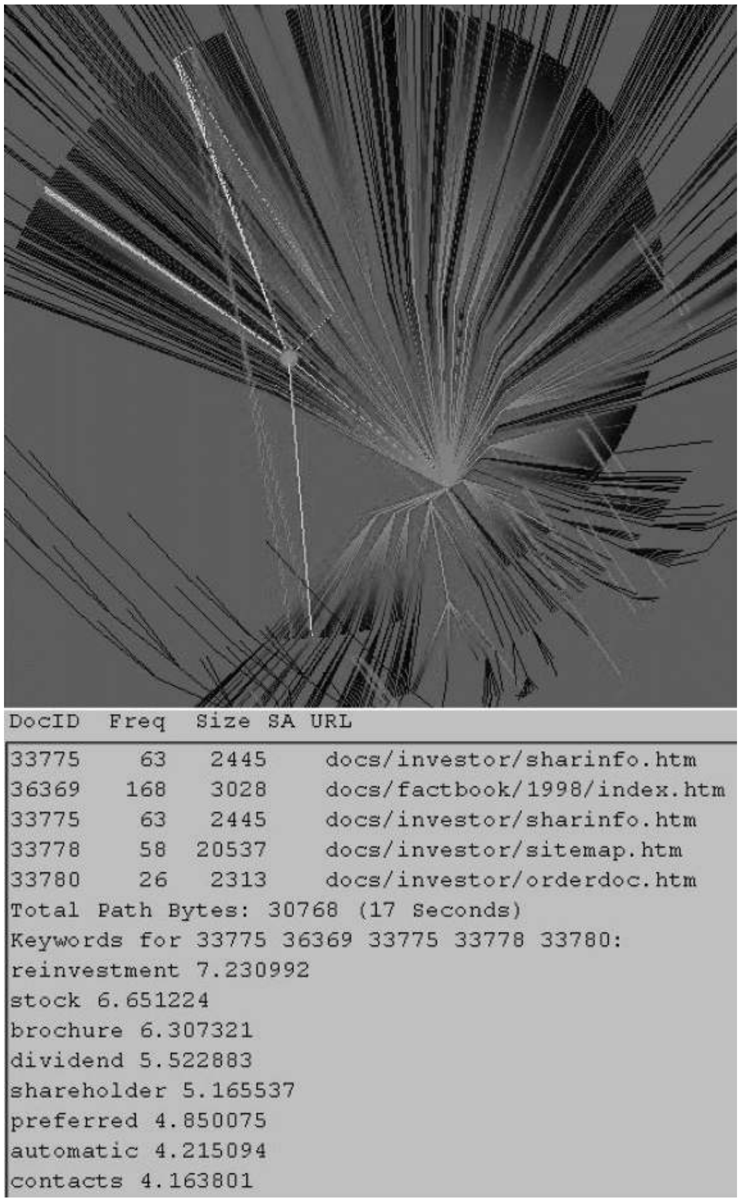
Visualization is also used for inferential analysis of WIMP and Web log files [Chi et al. 2000; Cugini and Scholtz 1999; Guzdial et al. 1994; Hochheiser and Shneiderman 2001]. It enables evaluators to filter, manipulate, and render log file data in a way that ideally facilitates analysis. Guzdial et 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 in order to gain an understanding of human factors issues related to visitation patterns. This approach combines the simultaneous display of a large number of individual datapoints (e.g., URLs requested vs. 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 8) and actual Web usage captured in server log files. This approach maps a Web site into a three-dimensional surface representing the hyperlinks (see the top part of Figure 3). 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 3. 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 4 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 2-D layout of the site where adjacent nodes are placed closer together than nonadjacent 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.

4.2.5. Automating Usability Testing Methods: Analysis Support—Discussion. Table IV summarizes log file analysis methods discussed in this section. Although the techniques vary widely on the four assessment criteria (effectiveness, ease of



```
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
```

Fig. 3. 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.

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); however, they require additional effort and learning time over

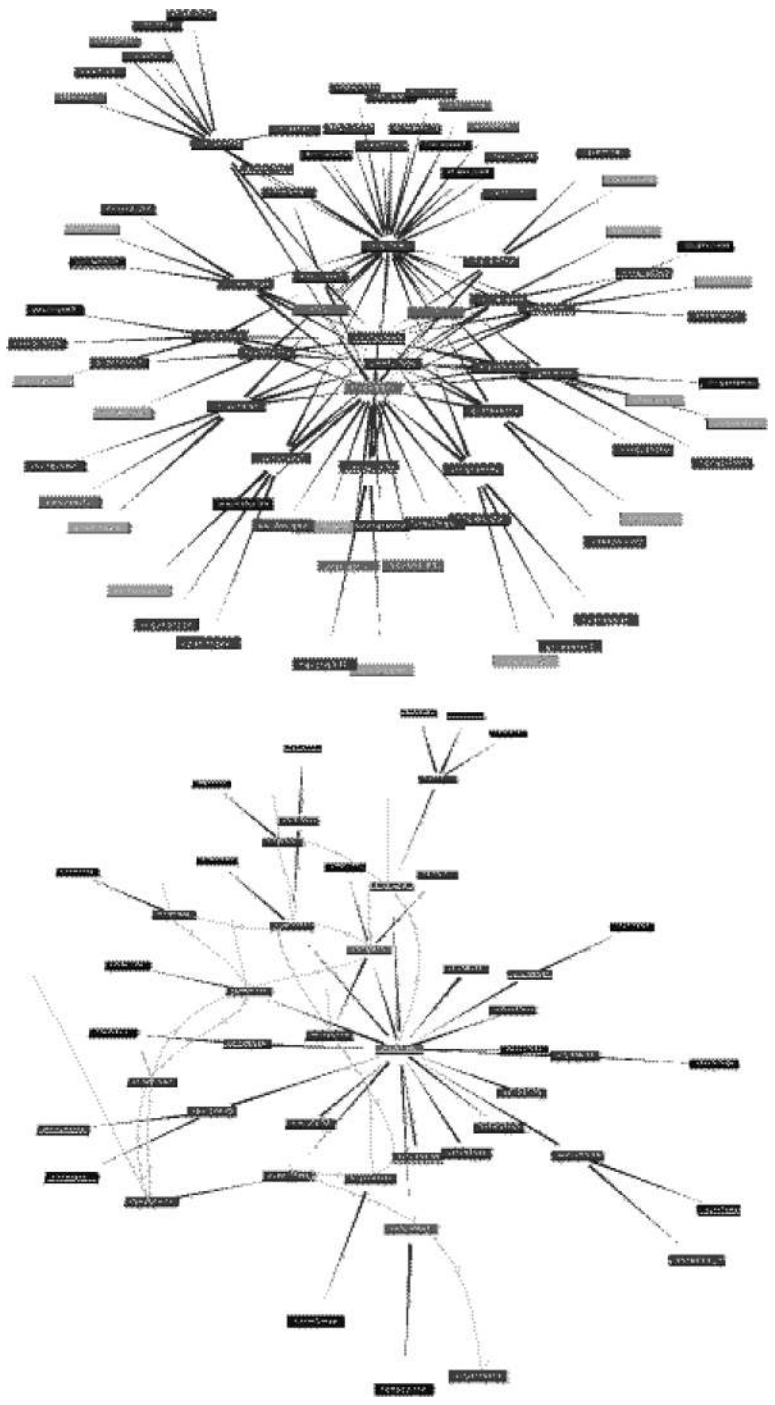


Fig. 4. VISVIP visualization [Cugini and Scholtz 1999] of a Web site (top part). The bottom part of the figure displays a usage path (series of directed lines on the left site) laid over a site. The top and bottom figures are for two different sites. Reprinted with permission of the authors.

Table V. Synopsis of Automated Capture Support for Inspection Methods[a]

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

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

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 prespecified 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.

5. AUTOMATING 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 nonautomated 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.

5.1. Automating Inspection Methods: Capture Support

Table V summarizes capture support for inspection methods, namely, a system

Table VI. Synopsis of Automated Analysis Support for Inspection Methods[a]

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	

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

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].

5.2. Automating Inspection Methods: Analysis Support

Table VI 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; we denote this with a blank entry in the effort column. We discuss support available for WIMP and Web UIs separately.

5.2.1. Automating Inspection Methods: Analysis Support—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, whereas 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. It 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. User studies demonstrated that tool results can be used to decrease screen search time and ultimately to improve screen layout. AIDE (semiautomated 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 to 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.

5.2.2. Automating Inspection Methods: Analysis Support—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 is less than three levels). (HyperAT also supports inferential analysis of server log files similar to other log file analysis techniques; see Section 4.2.) Gentler [Thimbleby 1997] provides similar structural analysis but focuses on maintenance of existing sites rather than the 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 attributes, 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 in order 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 as depicted in Figure 5. It currently does not provide suggestions for improving scanning paths.

5.2.3. Automating Inspection Methods: Analysis Support—Discussion. Table VI 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

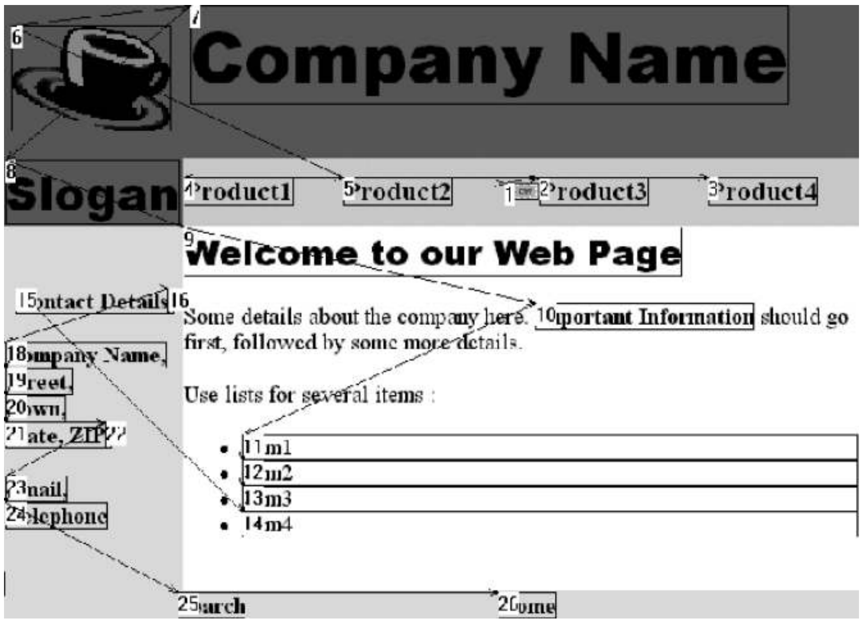


Fig. 5. The Design Advisor [Faraday 2000] superimposes a scanning path on the Web page. The numbers indicate the order in which elements will be scanned. Reproduced with permission of the author.

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 straightforward 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 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 1999; Fogg et al. 2000], since credibility affects usability and vice versa. This work presents a survey of over 1,400 Web users as well as an empirical study which indicated that typographical errors, ads, broken links, and other aspects have an impact on 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

Table VII. Synopsis of Automated Critique Support for Inspection Methods[a]

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	

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

special Web browser for use, but is easy to use, learn, and apply to basic Web pages (i.e., pages that do not 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.

We are currently developing a methodology for deriving Web design guidelines directly from sites that have been assessed by human judges [Ivory et al. 2000, 2001; Ivory 2001]. We have identified a number of Web page measures having to do with page composition (e.g., number of words, links, and images), page formatting (e.g., emphasized text, text positioning, and text clusters), and overall page characteristics (e.g., page size and download speed). Empirical studies have shown that we can predict with high accuracy whether a Web page will be rated favorably based on key metrics. Future work will identify profiles of highly rated pages that can be used to help designers improve Web UIs.

5.3. Automating Inspection Methods: Critique Support

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 are 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 VII provides a synopsis of automated critique methods discussed in the remainder of this section. All but one method, SYNOP, require minimal effort to employ; we denote this with a blank entry in the effort column. We discuss support available for WIMP and Web UIs separately.

5.3.1. Automating Inspection Methods: Critique Support—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 UI 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 [1986] and 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) in order 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.

5.3.2. Automating Inspection Methods: Critique Support—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 similar to WebSAT (discussed in Section 5.2.2) 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, and 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 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 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. WebEval [Scapin et al. 2000] is one automated critique approach being developed to address this issue. Similar 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.

5.3.3. Automating Inspection Methods: Critique Support—Discussion. Table VII 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.

Table VIII. Synopsis of Automated Capture Support for Inquiry Methods[a]

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, or NetRaker)	WIMP, Web	IF

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

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 11 automated Web site analysis methods, including Bobby and Lift Online. The 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.

6. AUTOMATING INQUIRY METHODS

Similar 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 employ 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.

6.1. Automating Inquiry Methods: Capture Support

Table VIII 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 into a WIMP UI to facilitate the response capture process. Typically dialog boxes prompt users for subjective input and process responses (e.g., saves data to a file or emails 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 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] for WIMP interfaces and WAMMI (Website analysis and measurement inventory) [Kirakowski and Claridge 1998] for Web interfaces. NetRaker's [2000] usability research tools enable evaluators to create custom HTML questionnaires and usability tests via a template interface and 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 nonautomated 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.

7. AUTOMATING ANALYTICAL MODELING METHODS

Analytical modeling complements traditional evaluation techniques such as 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 has been developed and supports 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. Three methods discussed in this section, GOMS analysis [John and Kieras 1996], CTA [May and Barnard 1994], and 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. Our 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.

Table IX. Synopsis of Automated Analysis Support for Analytical Modeling Methods[a]

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)		
Conduct a cognitive analysis of an interface and generate predictions (CTA)	WIMP	M
Method Type: Programmable User Models—write program that acts as a user (1 method)		
UE Method	UI	Effort
Program architecture to mimic user interaction with an interface (PUM)	WIMP	M

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

Automation has been predominately used to analyze task completion (e.g., execution and learning time) within WIMP UIs. Analytical modeling inherently supports automated analysis. Our 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. [1993]. 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 are discussed in Section 8.

7.1. Automating Analytical Modeling Methods: Analysis Support

Table IX 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.

7.1.1. Automating Analytical Modeling Methods: Analysis Support—WIMP UIs. The GOMS family of analytical 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 et al. [1993], (CMN-GOMS) 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 vs. a high-level procedure) and in the support for alternative task completion methods and support for single goals versus multiple simultaneous goals.

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 8). 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.

USAGE[3] (the UIDE System for semiautomated 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 8). 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 from 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 8), 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 as 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), the model can then be used to generate quantitative performance predictions similar 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.

7.1.2. Automating Analytical Modeling Methods: Analysis Support—Discussion. Table IX summarizes automated analysis methods discussed in this section. Analytical modeling approaches enable the evaluator to produce relatively inexpensive results to inform design choices. GOMS 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

[3] This is not to be confused with the UsAGE log file capture and analysis tool discussed in Section 4.

[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, our survey did not uncover any approaches that address this gap in Web site evaluation. Approaches such as 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 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.

8. AUTOMATING SIMULATION METHODS

Simulation complements traditional UE methods and, like analytical modeling, can be viewed as inherently supporting automated analysis. Using models of the user and/or the interface design, these approaches 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 simulations with different parameters in order 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.

8.1. Automating Simulation Methods: Capture Support

Table X 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; these 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

Table X. Synopsis of Automated Capture Support for Simulation Methods[a]

Method Class: Simulation		
Automation Type: Capture		
Method Type: Genetic Algorithm Modeling—mimic novice user interaction (1 method)		
UE Method	UI	Effort
Generate and capture usage traces representing novice users (Kasik and George [1996])	WIMP	
Method Type: Information Scent Modeling—mimic Web site navigation (1 method)		
UE Method	UI	Effort
Generate and capture navigation paths for information-seeking tasks (Chi et al. [2000])	Web	M

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

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., specified pages) as input to the simulator. The simulation 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 specified 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 specified pages.

The authors use these usage paths as input to the Dome Tree visualization methodology, an inferential log file analysis approach discussed in Section 4. The authors compare actual and simulated navigation paths for Xerox's corporate site and discover 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 choices. Hence, this approach may enable only crude approximations of user behavior for sites with complex pages.

8.1.1. Automating Simulation Methods: Capture Support—Discussion. Table X 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 [1976] 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

Table XI. Synopsis of Automated Analysis Support for Simulation Methods[a]

Method Class: Simulation		
Automation Type: Analysis		
Method Type: Petri Net Modeling—mimic user interaction from usage data (1 method)		
UE Method	UI	Effort
Construct Petri nets from usage data and analyze problem solving (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

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M).

evaluator must manually analyze the execution of each trace in order to identify problems. The authors propose future work to automatically verify that a trace produces 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. [2000] 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.

8.2. Automating Simulation Methods: Analysis Support

Table XI provides a synopsis of the automated analysis methods discussed in the remainder of this section. We consider methods for WIMP and Web UIs separately.

8.2.1. Automating Simulation Methods: Analysis Support—WIMP UIs. AMME [Rauterberg and Aeppili 1995] (see Section 4.1) 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) in order to develop a simulator. Hence, AMME is more accurate and 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. The behavior complexity measure has been shown to correlate 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 [1998] provide a detailed discussion of computational cognitive architectures and an overview of many approaches, including five that we discuss: ACT-R (adaptive control of thought) [Anderson 1990, 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, we also consider 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]. Rather than describe each method individually, we summarize the major characteristics of these simulation methods in Table XII and discuss them below.

Modeled Tasks. The models we surveyed 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) whereas others incorporate perceptual and motor processing as well (EPIC, ICS, HOS, Soar, GLEAN, CCT).

Component Processing. Task execution is modeled as serial processing (ACT-R, GLEAN, CCT), parallel processing (EPIC, ICS, Soar), or semiparallel 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 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 (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 do so by modeling UI interaction and all components (perception, cognition, and motor) 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 employs 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 demonstrated that predictions for telephone operator and menu searching tasks closely match observed data. EPIC

Table XII. Characteristics of WIMP Simulation Methods Based on a Variation of the MHP

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
semiparallel	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

and all of the other methods require considerable learning time and effort to employ. They are also applicable to a wide range of WIMP UIs.

8.2.2. Automating Simulation Methods: Analysis Support—Web UIs. Our 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 to 600 unique pages) to collect Web site data. These data are 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 5). 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.

8.2.3. Automating Simulation Methods: Analysis Support—Discussion. Table XI summarizes automated analysis methods discussed in this section. Unlike most evaluation approaches, simulation can be

Table XIII. Synopsis of Automated Analysis Support for Usability Testing Methods[a]

Method Class: Usability Testing		
Automation Type: Analysis		
Method Type: Log File Analysis—analyze usage data (19 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, 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 Web	IF IF

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M). This is a repetition of Table IV.

employed 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].

9. EXPANDING EXISTING APPROACHES TO AUTOMATING USABILITY EVALUATION METHODS

Automated usability evaluation methods have many potential benefits, including reducing the costs of nonautomated methods, aiding in comparisons between alternative designs, and improving consistency in evaluation results. We have studied numerous methods that support automation. Based on the methods surveyed, it is our opinion that research to further develop log file analysis, guideline review, analytical modeling, and simulation techniques could result in several promising automated techniques as discussed in more detail below. Ivory [2001] provides a more detailed discussion of techniques that merit further research, including approaches based on performance evaluation of computer systems.

9.1. Expanding Log File Analysis Approaches

Our survey showed log file analysis to be a viable methodology for automated analysis of usage data. Table XIII summarizes current approaches to log file analysis. We propose the following three ways to expand and improve on these approaches.

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. We discussed 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 8.1), that automatically generate plausible usage data. The genetic algorithm approach determines user behavior during deviation points in an expert user script, and 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.

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 4), 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.

Augmenting task-based pattern-matching approaches with guidelines in order 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. In addition, since USINE already reports substantial analysis data, these data could be compared to usability guidelines in order to support automated critique.

9.2. Expanding Guideline Review Approaches

Several guideline review methods for analysis of WIMP interfaces (see Table XIV) could be augmented with guidelines to support automated critique. For example, AIDE [Sears 1995] (discussed in Section 5) 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 XIV and XV), existing approaches only cover a small fraction of usability aspects [Brajnik 2000] and have not been empirically validated. We are developing a methodology for deriving Web design guidelines directly from sites that have been assessed by human judges

Table XIV. Synopsis of Automated Analysis Support for Inspection Methods[a]

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	

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M). This is a repetition of Table VI.

Table XV. Synopsis of Automated Critique Support for Inspection Methods[a]

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	

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M). This is a repetition of Table VII.

[Ivory et al. 2000, 2001; Ivory 2001]. We have identified a number of Web page measures having to do with page composition (e.g., number of words, links, and images), page formatting (e.g., emphasized text, text positioning, and text clusters), and overall page characteristics (e.g., page size and download speed). Empirical studies have shown that we can predict with high accuracy whether a Web page will be rated favorably based on key metrics. Future work will identify profiles of highly rated pages that can be used to help designers improve Web UIs.

9.3. Expanding Analytical Modeling Approaches

Our survey showed that evaluation within a user interface development environment (UIDE) is a promising approach for automated analysis via analytical modeling. Table XVI summarizes current approaches to analytical modeling. We propose to augment UIDE analysis methods, such as CRITIQUE and GLEAN, 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 these user interface development environments is an open research problem.

Cognitive task analysis [May and Barnard 1994] provides some insight for

Table XVI. Synopsis of Automated Analysis Support for Analytical Modeling Methods[a]

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)		
Conduct a cognitive analysis of an interface and generate predictions (CTA)	WIMP	M
Method Type: Programmable User Models—rite program that acts as a user (1 method)		
UE Method	UI	Effort
Program architecture to mimic user interaction with an interface (PUM)	WIMP	M

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M). This is a repetition of Table IX.

Table XVII. Synopsis of Automated Analysis Support for Simulation Methods[a]

Method Class: Simulation		
Automation Type: Analysis		
Method Type: Petri Net Modeling—imic user interaction from usage data (1 method)		
UE Method	UI	Effort
Construct Petri nets from usage data and analyze problem solving (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

[a]The effort level for each method is represented as: minimal (blank), formal (F), informal (I), and model (M). This is a repetition of Table XI.

analyzing UIs outside 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 HCI, such as in the performance evaluation of computer systems [Ivory 2001; Jain 1991]; 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.

9.4. Expanding Simulation Approaches

Table XVII summarizes current approaches to simulation analysis. Our 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.

We propose using several simulation techniques employed in the performance analysis of computer systems, in particular, trace-driven discrete-event simulation and Monte Carlo simulation [Ivory 2001; Jain 1991], to enable designers to perform what-if analyses with UIs. Trace-driven discrete-event simulations employ 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 relaxing the need for explicit task hierarchies or user models, although such models could still be employed. 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 of Monte Carlo simulation.

10. CONCLUSIONS

In this article we provided an overview of usability evaluation automation and presented a taxonomy for comparing various methods. We also presented an extensive survey of the use of automation in WIMP and Web interface evaluation, finding that automation is used in only 33% of methods surveyed. Of all the surveyed methods, only 29% are free from requirements of formal or informal interface use. All approaches that do not require formal or informal interface 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 nonautomated 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 nonautomated 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 new, effective automated techniques.

APPENDIX

A. AUTOMATION CHARACTERISTICS OF WIMP AND WEB INTERFACES

The following tables depict automation characteristics for WIMP and Web interfaces separately. We combined this information in Table I.

Table XVIII. Automation Support for 75 WIMP UE Methods[a]

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)			
Codiscovery Learning	F (1)			
Performance Measurement	F (1)	F (4)		
Log File Analysis			IFM (10)*	
Retrospective Testing	F (1)			
Remote Testing		IF (2)		
Inspection				
Guideline Review	IF (2)		(3)	M (5)†
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)			
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 (8)	
Petri Net Modeling			FM (1)	
Genetic Algorithm Modeling		(1)		
Automation Type				
Total	30	5	8	1
Percent	68%	11%	18%	2%

[a]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). Four software tools provide automation support for multiple method types: DRUM, performance measurement and log file analysis; AMME, log file analysis and Petri net modeling; KALDI, performance measurement, log file analysis, and remote testing; and UsAGE, performance measurement and log file analysis.

*Either formal or informal interface use is required. In addition, a model may be used in the analysis.

†Methods may or may not employ a model.

Table XIX. Descriptions of the WIMP UE Method Types Depicted in Table XVIII

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
Codiscovery Learning	two users collaborate
Performance Measurement	tester 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 colocated 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 violations of heuristics
Perspective-Based Inspection	expert conducts narrowly focused heuristic evaluation
Feature Inspection	expert evaluates product features
Formal Usability Inspection	expert conducts 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

Table XX. Automation Support for 57 Web UE Methods[a]

Method Class	Automation Type			
Method 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)			
Codiscovery Learning	F (1)			
Performance Measurement	F (1)	F (3)		
Log File Analysis			IFM (9)	
Retrospective Testing	F (1)			
Remote Testing		IF (3)		
Inspection				
Guideline Review	IF (4)		(5)	(6)
Cognitive Walkthrough	IF (2)			
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 (1)		
Self-Reporting Logs	IF (1)			
Screen Snapshots	IF (1)			
User Feedback	IF (1)			
Analytical Modeling				
No Methods Surveyed				
Simulation				
Information Proc. Modeling			M (1)	
Information Scent Modeling		M (1)		
Automation Type				
Total	26	4	3	1
Percent	76%	12%	9%	3%

[a]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). Two software tools provide automation support for multiple method types: Dome Tree visualization, log file analysis and information scent modeling; and WebVIP, performance measurement and remote testing.

Table XXI. Descriptions of the Web UE Method Types Depicted in Table XX

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
Codiscovery Learning	two users collaborate
Performance Measurement	tester 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 colocated 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 violations of heuristics
Perspective-Based Inspection	expert conducts narrowly focused heuristic evaluation
Feature Inspection	expert evaluates product features
Formal Usability Inspection	expert conducts 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	
No Methods Surveyed	
Simulation	
Information Proc. Modeling	mimic user interaction
Information Scent Modeling	mimic Web site navigation

ACKNOWLEDGMENTS

We thank the anonymous reviewers for invaluable guidance on improving our presentation of this material. We thank James Hom and Zhijun Zhang for allowing us to use their extensive archives of usability methods for this survey. We also thank Zhijun Zhang for participating in several interviews on usability evaluation. We thank Bonnie John and Scott Hudson for helping us locate information on GOMS and other simulation methods for this survey, and James Landay and Mark Newman for helpful feedback and data.

REFERENCES

ABELOW, D. 1993. Automating feedback on software product use. *CASE Trends December*, 15–17.

ADDY & ASSOCIATES. 2000. Dr. Watson version 4.0. Available at http://watson.addy.com/.

AHLBERG, C. AND SHNEIDERMAN, B. 1994. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of the Conference on Human Factors in Computing Systems* (Boston, MA, April), pp. 313–317. New York, NY: ACM Press.

AL-QAIMARI, G. AND MCROSTIE, D. 1999. KALDI: A computer-aided usability engineering tool for supporting testing and analysis of human-computer interaction. In J. Vanerdonckt and A. Puerta, Eds., *Proceedings of the Third International Conference on Computer-Aided Design of User Interfaces* (Louvain-la-Neuve, Belgium, October). Dordrecht, The Netherlands: Kluwer Academic Publishers.

ANDERSON, J. 1993. *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

ANDERSON, J. R. 1990. *The Adaptive Character of Thought*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

BACHELDOR, B. 1999. Push for performance. *Information Week September 20*, 18–20.

BALBO, S. 1995. Automatic evaluation of user interface usability: Dream or reality. In S. Balbo, Ed., *Proceedings of the Queensland Computer-Human Interaction Symposium* (Queensland, Australia, August). Bond University.

BALBO, S. 1996. ÉMA: Automatic analysis mechanism for the ergonomic evaluation of user interfaces. Tech. Rep. 96/44 (August), CSIRO Division of Information Technology. Available at http://www.cmis.csiro.au/sandrine.balbo/Ema/ema_tr/ema-tr.doc.

BARNARD, P. J. 1987. Cognitive resources and the learning of human-computer dialogs. In J. M. Carroll, Ed., *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, pp. 112–158. Cambridge, MA: MIT Press.

BARNARD, P. J. AND TEASDALE, J. D. 1991. Interacting cognitive subsystems: A systemic approach to cognitive-affective interaction and change. *Cognition and Emotion 5*, 1–39.

BAUMEISTER, L. K., JOHN, B. E., AND BYRNE, M. D. 2000. A comparison of tools for building GOMS models. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 1 (The Hague, The Netherlands, April), pp. 502–509. New York, NY: ACM Press.

BEARD, D. V., SMITH, D. K., AND DENELSBECK, K. M. 1996. Quick and dirty GOMS: A case study of computed tomography interpretation. *Human-Computer Interaction 11*, 2, 157–180.

BELLOTTI, V. 1988. Implications of current design practice for the use of HCI techniques. In *Proceedings of the HCI Conference on People and Computers IV* (Manchester, United Kingdom, September), pp. 13–34. Amsterdam, The Netherlands: Elsevier Science Publishers.

BODART, F., HENNEBERT, A. M., LEHEUREUX, J. M., AND VANDERDONCKT, J. 1994. Towards a dynamic strategy for computer-aided visual placement. In T. Catarci, M. Costabile, M. Levialdi, and G. Santucci, Eds., *Proceedings of the International Conference on Advanced Visual Interfaces* (Bari, Italy), pp. 78–87. New York, NY: ACM Press.

BORGES, J. A., MORALES, I., AND RODRIGUEZ, N. J. 1996. Guidelines for designing usable world wide web pages. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 2 (Vancouver, Canada, April), pp. 277–278. New York, NY: ACM Press.

BOWERS, N. 1996. Weblint: Quality assurance for the World Wide Web. In *Proceedings of the Fifth International World Wide Web Conference* (Paris, France, May). Amsterdam, The Netherlands: Elsevier Science Publishers. Available at http://www5conf.inria.fr/fich_html/papers/P34/Overview.html.

BRAJNIK, G. 2000. Automatic web usability evaluation: Where is the limit? In *Proceedings of the Sixth Conference on Human Factors & the Web* (Austin, TX, June). Available at http://www.tri.sbc.com/hfweb/brajnik/hfweb-brajnik.html.

BYRNE, M. D., JOHN, B. E., WEHRLE, N. S., AND CROW, D. C. 1999. The tangled web we wove: A taxonomy of WWW use. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 1 (Pittsburgh, PA, May), pp. 544–551, New York, NY: ACM Press.

BYRNE, M. D., WOOD, S. D., SUKAVIRIYA, P. N., FOLEY, J. D., AND KIERAS, D. 1994. Automating interface evaluation. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 1 (April), pp. 232–237. New York, NY: ACM Press.

CARD, S. K., MORAN, T. P., AND NEWELL, A. 1983. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

CENTERLINE. 1999. QC/Replay. Available at http://www.centerline.com/productline/qcreplay/qcreplay.html.

CHAK, A. 2000. Usability tools: A useful start. *Web Techniques* (2000), August, 18–20. Available at http://www.webtechniques.com/archives/2000/08/stratrevu/.

CHI, E. H., PIROLLI, P., AND PITKOW, J. 2000. The scent of a site: A system for analyzing and predicting information scent, usage, and usability of a web site. In *Proceedings of the Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April), pp. 161–168. New York, NY: ACM Press.

CLARK, D. AND DARDAILLER, D. 1999. Accessibility on the web: Evaluation & repair tools to make it possible. In *Proceedings of the CSUN Technology and Persons with Disabilities Conference* (Los Angeles, CA, March). Available at http://www.dinf.org/csun_99/session0030.html.

COMBER, T. 1995. Building usable web pages: An HCI perspective. In R. Debreceny and A. Ellis, Eds., *Proceedings of the First Australian World Wide Web Conference* (Ballina, Australia, April), pp. 119–124. Ballina, Australia: Norsearch. Available at http://www.scu.edu.au/sponsored/ausweb/ausweb95/papers/hypertext/comber/.

COOPER, M. 1999. Universal design of a web site. In *Proceedings of the CSUN Technology and Persons with Disabilities Conference* (Los Angeles, CA, March). Available at http://www.dinf.org/csun_99/session0030.html.

COUTAZ, J. 1995. Evaluation techniques: Exploring the intersection of HCI and software engineering. In R. N. Taylor and J. Coutaz, Eds., *Software Engineering and Human-Computer Interaction*, Lecture Notes in Computer Science, pp. 35–48. Heidelberg, Germany: Springer-Verlag.

CUGINI, J. AND SCHOLTZ, J. 1999. VISVIP: 3D visualization of paths through web sites. In *Proceedings of the International Workshop on Web-Based Information Visualization* (Florence, Italy, September), pp. 259–263. Institute of Electrical and Electronics Engineers.

de Haan, G., van der Veer, G. C., and van Vliet, J. C. 1992. Formal modelling techniques in human-computer interaction. In G. C. van der Veer, S. Bagnara, and G. A. M. Kempen, Eds., *Cognitive Ergonomics*: *Contributions from Experimental Psychology*, Theoretical Issues, pp. 27–67. Amsterdam, The Netherlands: Elsevier Science Publishers.

de Souza, F. and Bevan, N. 1990. The use of guidelines in menu interface design: Evaluation of a draft standard. In G. Cockton, D. Diaper, and B. Shackel, Eds., *Proceedings of the Third IFIP TC13 Conference on Human-Computer Interaction* (Cambridge, UK, August), pp. 435–440. Amsterdam, The Netherlands: Elsevier Science Publishers.

Detweiler, M. C. and Omanson, R. C. 1996. Ameritech web page user interface standards and design guidelines. Ameritech Corporation, Chicago, IL. Available at http://www.ameritech.com/corporate/testtown/library/standard/web_guidelines/index.html.

Dix, A., Finlay, J., Abowd, G., and Beale, R. 1998. *Human-Computer Interaction* (second ed.). Prentice Hall, Upper Saddle River, NJ.

Drott, M. C. 1998. Using web server logs to improve site design. In *Proceedings of the 16th International Conference on Systems Documentation* (Quebec, Canada, September), pp. 43–50. New York, NY: ACM Press.

Etgen, M. and Cantor, J. 1999. What does getting WET (web event-logging tool) mean for web usability. In *Proceedings of the Fifth Conference on Human Factors & the Web* (Gaithersburg, MD, June). Available at http://www.nist.gov/itl/div894/vvrg/hfweb/proceedings/etgen-cantor/index.html.

Faraday, P. 2000. Visually critiquing web pages. In *Proceedings of the Sixth Conference on Human Factors & the Web* (Austin, TX, June). Available at http://www.tri.sbc.com/hfweb/faraday/faraday.htm.

Faraday, P. and Sutcliffe, A. 1998. Providing advice for multimedia designers. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 1 (Los Angeles, CA, April), pp. 124–131. New York, NY: ACM Press.

Farenc, C. and Palanque, P. 1999. A generic framework based on ergonomics rules for computer aided design of user interface. In J. Vanderdonckt and A. Puerta, Eds., *Proceedings of the Third International Conference on Computer-Aided Design of User Interfaces* (Louvain-la-Neuve, Belgium, October). Dordrecht, The Netherlands: Kluwer Academic Publishers.

Farenc, C., Liberati, V., and Barthet, M.-F. 1999. Automatic ergonomic evaluation: What are the limits. In *Proceedings of the Third International Conference on Computer-Aided Design of User Interfaces* (Louvain-la-Neuve, Belgium, October). Dordrecht, The Netherlands: Kluwer Academic Publishers.

Fogg, B. J. 1999. What variables affect web credibility? Available at http://www.webcredibility.org/variables_files/v3_document.htm.

Fogg, B. J., Marshall, J., Osipovich, A., Varma, C., Laraki, O., Fang, N., Pavi, J., Rangnekar, A., Shon, J., Swani, P., and Treinen, M. 2000. Elements that affect web credibility: Early results from a self-report study. In *Proceedings of the Conference on Human Factors in Computing Systems*, number Extended Abstracts, 287–288. The Hague, The Netherlands. New York, NY: ACM Press. Available at http://www.webcredibility.org/webCredEarly-Results.ppt.

Fuller, R. and de Graaff, J. J. 1996. Measuring user motivation from server log files. In *Proceedings of the Second Conference on Human Factors & the Web* (Redmond, WA, October). Available at http://www.microsoft.com/usability/ webconf/fuller/fuller.htm.

Glenn, F. A., Schwartz, S. M., and Ross, L. V. 1992. Development of a human operator simulator version v (HOS-V): Design and implementation. U.S. Army Research Institute for the Behavioral and Social Sciences, PERI-POX, Alexandria, VA.

Guzdial, M., Santos, P., Badre, A., Hudson, S., and Gray, M. 1994. Analyzing and visualizing log files: A computational science of usability. GVU Center TR GIT-GVU-94-8, Georgia Institute of Technology. Available at http://www.cc.gatech.edu/gvu/reports/1994/abstracts/94-08.html.

Hammontree, M. L., Hendrickson, J. J., and Hensley, B. W. 1992. Integrated data capture and analysis tools for research and testing on graphical user interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems* (Monterey, CA, May), pp. 431–432. New York, NY: ACM Press.

Harper, B. and Norman, K. 1993. Improving user satisfaction: The questionnaire for user satisfaction interaction version 5.5. In *Proceedings of the First Annual Mid-Atlantic Human Factors Conference* (Virginia Beach, VA), pp. 224–228.

Hartson, H. R., Castillo, J. C., Kelsa, J., and Neale, W. C. 1996. Remote evaluation: The network as an extension of the usability laboratory. In M. J. Tauber, V. Bellotti, R. Jeffries, J. D. Mackinlay, and J. Nielsen, Eds., *Proceedings of the Conference on Human Factors in Computing Systems* (Vancouver, Canada, April), pp. 228–235. New York, NY: ACM Press.

Helfrich, B. and Landay, J. A. 1999. QUIP: quantitative user interface profiling. Unpublished manuscript. Available at http://home.earthlink.net/~bhelfrich/quip/index.html.

Hochheiser, H. and Shneiderman, B. 2001. Using interactive visualizations of WWW log data to characterize access patterns and inform site design. *Journal of American Society for Information Science and Technology 52*, 4 (February), 331–343.

HOM, J. 1998. The usability methods toolbox. Available at http://www.best.com/~jthom/usability/usable.htm.

HUDSON, S. E., JOHN, B. E., KNUDSEN, K., AND BYRNE, M. D. 1999. A tool for creating predictive performance models from user interface demonstrations. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology* (Asheville, NC, November), pp. 92–103. New York, NY: ACM Press.

HUMAN FACTORS ENGINEERING. 1999. Usability evaluation methods. Available at http://www.cs.umd.edu/~zzj/UsabilityHome.html.

INTERNATIONAL STANDARDS ORGANIZATION. 1999. Ergonomic requirements for office work with visual display terminals, part 11: Guidance on usability. Available at http://www.iso.ch/iso/en/catalogueDetailPage.catalogueDetail? CSNumber=16883&ICS1=13&ICS2=180&ICS3=.

IVORY, M. Y. 2001. An empirical foundation for automated web interface evaluation. PhD Thesis, University of California, Berkeley, Computer Science Division.

IVORY, M. Y., SINHA, R. R., AND HEARST, M. A. 2000. Preliminary findings on quantitative measures for distinguishing highly rated information-centric web pages. In *Proceedings of the Sixth Conference on Human Factors & the Web* (Austin, TX, June). Available at http://www.tri.sbc.com/hfweb/ivory/paper.html.

IVORY, M. Y., SINHA, R. R., AND HEARST, M. A. 2001. Empirically validated web page design metrics. In *Proceedings of the Conference on Human Factors in Computing Systems* (Seattle, WA, March), pp. 53–60. New York, NY: ACM Press.

JAIN, R. 1991. *Human-Computer Interaction*. Wiley-Interscience, New York, NY.

JEFFRIES, R., MILLER, J. R., WHARTON, C., AND UYEDA, K. M. 1991. User interface evaluation in the real world: A comparison of four techniques. In *Proceedings of the Conference on Human Factors in Computing Systems* (New Orleans, LA, April), pp. 119–124. New York, NY: ACM Press.

JIANG, J., MURPHY, E., AND CARTER, L. 1993. Computer-human interaction models (CHIMES): Revision 3. Tech. Rep. DSTL-94-008 (May), National Aeronautics and Space Administration.

JOHN, B. E. AND KIERAS, D. E. 1996. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction 3*, 4, 320–351.

KASIK, D. J. AND GEORGE, H. G. 1996. Toward automatic generation of novice user test scripts. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 1 (Vancouver, Canada, April), pp. 244–251. New York, NY: ACM Press.

KIERAS, D. AND POLSON, P. G. 1985. An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies 22*, 4, 365–394.

KIERAS, D. E., WOOD, S. D., ABOTEL, K., AND HORNOF, A. 1995. GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. In *Proceedings of the Eighth ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, November), pp. 91–100. New York, NY: ACM Press.

KIERAS, D. E., WOOD, S. D., AND MEYER, D. E. 1997. Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *ACM Transactions on Computer-Human Interaction 4*, 3 (September), 230–275.

KIM, W. C. AND FOLEY, J. D. 1993. Providing high-level control and expert assistance in the user interface presentation design. In S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White, Eds., *Proceedings of the Conference on Human Factors in Computing Systems* (Amsterdam, The Netherlands, April), pp. 430–437. New York, NY: ACM Press.

KIRAKOWSKI, J. AND CLARIDGE, N. 1998. Human centered measures of success in web site design. In *Proceedings of the Fourth Conference on Human Factors & the Web* (Basking Ridge, NJ, June). Available at http://www.research.att.com/conf/hfweb/proceedings/kirakowski/index.html.

LAIRD, J. E. AND ROSENBLOOM, P. 1996. The evolution of the Soar cognitive architecture. In D. M. Steier and T. M. Mitchell, Eds., *Mind Matters: A Tribute to Allen Newell*, pp. 1–50. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

LARSON, K. AND CZERWINSKI, M. 1998. Web page design: Implications of memory, structure and scent for information retrieval. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 1 (Los Angeles, CA, April), pp. 25–32. New York, NY: ACM Press.

LECEROF, A. AND PATERNÒ, F. 1998. Automatic support for usability evaluation. *IEEE Transactions on Software Engineering 24*, 10 (October), 863–888.

LEE, K. 1997. Motif FAQ. Available at http://www-bioeng.ucsd.edu/~fvetter/misc/Motif-FAQ.txt.

LEVINE, R. 1996. Guide to web style. Sun Microsytems. Available at http://www.sun.com/styleguide/.

LEWIS, C., POLSON, P. G., WHARTON, C., AND RIEMAN, J. 1990. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems* (Seattle, WA, April), pp. 235–242. New York, NY: ACM Press.

LOWGREN, J. AND NORDQVIST, T. 1992. Knowledge-based evaluation as design support for graphical user interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems* (Monterey, CA, May), pp. 181–188. New York, NY: ACM Press.

LYNCH, G., PALMITER, S., AND TILT, C. 1999. The max model: A standard web site user model. In *Proceedings of the Fifth Conference on Human Factors & the Web* (Gaithersburg, MD, June). Available at http://www.nist.gov/itl/div894/vvrg/hfweb/ proceedings/lynch/index.html.

LYNCH, P. J. AND HORTON, S. 1999. *Web Style Guide*: *Basic Design Principles for Creating Web Sites*. Yale University Press. Available at http://info.med.yale.edu/caim/manual/.

MACLEOD, M. AND RENGGER, R. 1993. The development of DRUM: A software tool for video-assisted usability evaluation. In *Proceedings of the HCI Conference on People and Computers VIII* (Loughborough, UK, September), pp. 293–309. Cambridge University Press, Cambridge, UK.

MAHAJAN, R. AND SHNEIDERMAN, B. 1997. Visual & textual consistency checking tools for graphical user interfaces. *IEEE Transactions on Software Engineering 23*, 11 (November), 722–735.

MAY, J. AND BARNARD, P. J. 1994. Supportive evaluation of interface design. In C. Stary, Ed., *Proceedings of the First Interdisciplinary Workshop on Cognitive Modeling and User Interface Design* (Vienna, Austria, December).

MERCURY INTERACTIVE. 2000. Winrunner. Available at http://www-svca.mercuryinteractive.com/products/winrunner/.

MOLICH, R., BEVAN, N., BUTLER, S., CURSON, I., KINDLUND, E., KIRAKOWSKI, J., AND MILLER, D. 1998. Comparative evaluation of usability tests. In *Proceedings of the UPA Conference* (Washington, DC, June), pp. 189–200. Usability Professionals' Association, Chicago, IL.

MOLICH, R., THOMSEN, A. D., KARYUKINA, B., SCHMIDT, L., EDE, M., VAN OEL, W., AND ARCURI, M. 1999. Comparative evaluation of usability tests. In *Proceedings of the Conference on Human Factors in Computing Systems* (Pittsburgh, PA, May), pp. 83–86. New York, NY: ACM Press.

MORAN, T. P. 1981. The command language grammar: A representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies 15*, 1, 3–50.

MORAN, T. P. 1983. Getting into a system: External-internal task mapping analysis. In *Proceedings of the Conference on Human Factors in Computing Systems* (Boston, MA, December), pp. 45–49. New York, NY: ACM Press.

NETRAKER. 2000. The NetRaker suite. Available at http://www.netraker.com/info/applications/index.asp.

NIELSEN, J. 1993. *Usability Engineering*. Boston, MA: Academic Press.

OLSEN, JR., D. R. 1992. *User Interface Management Systems: Models and Algorithms*. San Mateo, CA: Morgan Kaufman Publishers, Inc.

OLSEN, JR., D. R. AND HALVERSEN, B. W. 1988. Interface usage measurements in a user interface management system. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software* (Alberta, Canada, October), pp. 102–108. New York, NY: ACM Press.

OPEN SOFTWARE FOUNDATION. 1991. *OSF/Motif Style Guide*. Number Revision 1.1 (for OSF/Motif release 1.1). Englewood Cliffs, NJ: Prentice Hall.

PALANQUE, P., FARENC, C., AND BASTIDE, R. 1999. Embedding ergonomic rules as generic requirements in the development process of interactive software. In A. Sasse and C. Johnson, Eds., *Proceedings of IFIP TC13 Seventh International Conference on Human-Computer Interaction* (Edinburgh, Scotland, August). Amsterdam, The Netherlands: IOS Press.

PARUSH, A., NADIR, R., AND SHTUB, A. 1998. Evaluating the layout of graphical user interface screens: Validation of a numerical, computerized model. *International Journal of Human Computer Interaction 10*, 4, 343–360.

PATERNÒ, F. AND BALLARDIN, G. 1999. Model-aided remote usability evaluation. In A. Sasse and C. Johnson, Eds., *Proceedings of the IFIP TC13 Seventh International Conference on Human-Computer Interaction* (Edinburgh, Scotland, August), pp. 434–442. Amsterdam, The Netherlands: IOS Press.

PATERNÒ, F., MANCINI, C., AND MENICONI, S. 1997. ConcurTaskTrees: Diagrammatic notation for specifying task models. In *Proceedings of the IFIP TC13 Sixth International Conference on Human-Computer Interaction* (Sydney, Australia, July), pp. 362–369. Sydney, Australia: Chapman and Hall.

PAYNE, S. J. AND GREEN, T. R. G. 1986. Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction 2*, 93–133.

PECK, V. A. AND JOHN, B. E. 1992. Browser-soar: A computational model of a highly interactive task. In *Proceedings of the Conference on Human Factors in Computing Systems* (Monterey, CA, May), pp. 165–172. New York, NY: ACM Press.

PETRI, C. A. 1973. Concepts of net theory. In *Mathematical Foundations of Computer Science*: *Proceedings of the Symposium and Summer School* (High Tatras, Czechoslovakia, September), pp. 137–146. Mathematical Institute of the Slovak Academy of Sciences.

PEW, R. W. AND MAVOR, A. S., EDS. 1998. *Modeling Human and Organizational Behavior*: *Application to Military Simulations*. Washington, DC: National Academy Press. Available at http://books.nap.edu/html/model.

POLK, T. A. AND ROSENBLOOM, P. S. 1994. Task-independent constraints on a unified theory of cognition. In F. Boller and J. Grafman, Eds., *Handbook of Neuropsychology, Volume 9*. Amsterdam, The Netherlands: Elsevier Science Publishers.

RATNER, J., GROSE, E. M., AND FORSYTHE, C. 1996. Characterization and assessment of HTML style guides. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 2 (Vancouver, Canada, April), pp. 115–116. New York, NY: ACM Press.

RAUTERBERG, M. 1995. From novice to expert decision behaviour: A qualitative modeling approach with Petri nets. In Y. Anzai, K. Ogawa, and H. Mori, Eds., *Symbiosis of Human and Artifact*: *Human and Social Aspects of Human-Computer Interaction*, Volume 20B of *Advances in Human Factors/Ergonomics* (1995), pp. 449–454. Amsterdam, The Netherlands: Elsevier Science Publishers.

RAUTERBERG, M. 1996a. How to measure and to quantify usability of user interfaces. In A. Özok and G. Salvendy, Eds., *Advances in Applied Ergonomics* (1996), pp. 429–432. West Lafayette, IN: USA Publishing.

RAUTERBERG, M. 1996b. A Petri net based analyzing and modeling tool kit for logfiles in human-computer interaction. In *Proceedings of Cognitive Systems Engineering in Process Control* (Kyoto, Japan, November), pp. 268–275. Kyoto University: Graduate School of Energy Science.

RAUTERBERG, M. AND AEPPILI, R. 1995. Learning in man-machine systems: the measurement of behavioural and cognitive complexity. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics* (Vancouver, BC, October), pp. 4685–4690. Institute of Electrical and Electronics Engineers.

REISNER, P. 1984. Formal grammar as a tool for analyzing ease of use: Some fundamental concepts. In J. C. Thomas and M. L. Schneider, Eds., *Human Factors in Computer Systems*, pp. 53–78. Norwood, NJ: Ablex Publishing Corp.

REITERER, H. 1994. A user interface design assistant approach. In K. Brunnstein and E. Raubold, Eds., *Proceedings of the IFIP 13th World Computer Congress*, Volume 2 (Hamburg, Germany, August), pp. 180–187. Amsterdam, The Netherlands: Elsevier Science Publishers.

RIEMAN, J., DAVIES, S., HAIR, D. C., ESEMPLARE, M., POLSON, P., AND LEWIS, C. 1991. An automated cognitive walkthrough. In *Proceedings of the Conference on Human Factors in Computing Systems* (New Orleans, LA, April), pp. 427–428. New York, NY: ACM Press.

SCAPIN, D., LEULIER, C., VANDERDONCKT, J., MARIAGE, C., BASTIEN, C., FARENC, C., PALANQUE, P., AND BASTIDE, R. 2000. A framework for organizing web usability guidelines. In *Proceedings of the Sixth Conference on Human Factors & the Web* (Austin, TX, June). Available at http://www.tri.sbc.com/hfweb/scapin/Scapin.html.

SCHOLTZ, J. AND LASKOWSKI, S. 1998. Developing usability tools and techniques for designing and testing web sites. In *Proceedings of the Fourth Conference on Human Factors & the Web* (Basking Ridge, NJ, June). Available at http://www.research.att.com/conf/hfweb/ proceedings/scholtz/index.html.

SCHWARTZ, M. 2000. Web site makeover. *Computerworld January 31*. Available at http://www.computerworld.com/home/print.nsf/all/000126e3e2.

SEARS, A. 1995. AIDE: A step toward metric-based interface development tools. In *Proceedings of the Eighth ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, November), pp. 101–110. New York, NY: ACM Press.

SERVICE METRICS. 1999. Service metrics solutions. Available at http://www.servicemetrics.com/solutions/solutionsmain.asp.

SHNEIDERMAN, B. 1998. *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (third ed.). Reading, MA: Addison-Wesley.

SIOCHI, A. C. AND HIX, D. 1991. A study of computer-supported user interface evaluation using maximal repeating pattern analysis. In *Proceedings of the Conference on Human Factors in Computing Systems* (New Orleans, LA, April), pp. 301–305. New York, NY: ACM Press.

SMITH, S. L. 1986. Standards versus guidelines for designing user interface software. *Behaviour and Information Technology 5*, 1, 47–61.

SMITH, S. L. AND MOSIER, J. N. 1986. Guidelines for designing user interface software. Tech. Rep. ESD-TR-86-278, The MITRE Corporation, Bedford, MA 01730.

STEIN, L. D. 1997. The rating game. Available at http://stein.cshl.org/~lstein/rater/.

STREVELER, D. J. AND WASSERMAN, A. I. 1984. Quantitative measures of the spatial properties of screen designs. In B. Shackel, Ed., *Proceedings of the IFIP TC13 First International Conference on Human-Computer Interaction* (London, UK, September), pp. 81–89. Amsterdam, The Netherlands: North-Holland.

SULLIVAN, T. 1997. Reading reader reaction: A proposal for inferential analysis of web server log files. In *Proceedings of the Third Conference on Human Factors & the Web* (Boulder, CO, June). Available at http://www.research.att.com/conf/hfweb/conferences/denver3.zip.

TAUBER, M. J. 1990. ETAG: Extended task action grammar—A language for the description of the user's task language. In G. Cockton, D. Diaper, and B. Shackel, Eds., *Proceedings of the IFIP TC13 Third International Conference on Human-Computer Interaction* (Cambridge, UK, August), pp. 163–168. Amsterdam, The Netherlands: Elsevier Science Publishers.

THENG, Y. L. AND MARSDEN, G. 1998. Authoring tools: Towards continuous usability testing of web documents. In *Proceedings of the First International Workshop on Hypermedia Development* (Pittsburgh, PA, June). Available at http://www.eng.uts.edu.au/~dbl/HypDev/ht98w/YinLeng/HT98_YinLeng.html.

THIMBLEBY, H. 1997. Gentler: A tool for systematic web authoring. *International Journal of Human-Computer Studies 47*, 1, 139–168.

TULLIS, T. S. 1983. The formatting of alphanumeric displays: A review and analysis. *Human Factors 25*, 657–682.

UEHLING, D. L. AND WOLF, K. 1995. User action graphing effort (UsAGE). In *Proceedings of the Conference on Human Factors in Computing Systems* (Denver, CO, May). New York, NY: ACM Press.

USABLE NET. 2000. LIFT online. Available at http://www.usablenet.com.

VANDERDONCKT, J. AND GILLO, X. 1994. Visual techniques for traditional and multimedia layouts. In T. Catarci, M. Costabile, M. Levialdi, and G. Santucci, Eds., *Proceedings of the International Conference on Advanced Visual Interfaces* (Bari, Italy, June), pp. 95–104. New York, NY: ACM Press.

WEB ACCESSIBILITY INITIATIVE. 1999. Web content accessibility guidelines 1.0. World Wide Web Consortium, Geneva, Switzerland. Available at http://www.w3.org/TR/WAI-WEBCONTENT/.

WEB CRITERIA. 1999. Max, and the objective measurement of web sites. Available at http://www.webcriteria.com.

WEBTRENDS CORPORATION. 2000. Webtrends live. Available at http://www.webtrendslive.com/default.htm.

WHITEFIELD, A., WILSON, F., AND DOWELL, J. 1991. A framework for human factors evaluation. *Behaviour and Information Technology 10*, 1, 65–79.

WILLIAMS, K. E. 1993. Automating the cognitive task modeling process: An extension to GOMS for HCI. In *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 3 (Amsterdam, The Netherlands, April), pp. 182. New York, NY: ACM Press.

WORLD WIDE WEB CONSORTIUM. 2000. HTML validation service. Available at http://validator.w3.org/.

YOUNG, R. M., GREEN, T. R. G., AND SIMON, T. 1989. Programmable user models for predictive evaluation of interface designs. In *Proceedings of the Conference on Human Factors in Computing Systems* (Austin, TX, April), pp. 15–19. New York, NY: ACM Press.

ZACHARY, W., MENTEC, J.-C. L., AND RYDER, J. 1996. Interface agents in complex systems. In C. N. Ntuen and E. H. Park, Eds., *Human Interaction With Complex Systems: Conceptual Principles and Design Practice*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

ZAPHIRIS, P. AND MTEI, L. 1997. Depth vs. breadth in the arrangement of Web links. Available at http://www.otal.umd.edu/SHORE/bs04.

ZETTLEMOYER, L. S., ST. AMANT, R. S., AND DULBERG, M. S. 1999. IBOTS: Agent control through the user interface. In *Proceedings of the International Conference on Intelligent User Interfaces* (Redondo Beach, CA, Jan.), pp. 31–37. New York, NY: ACM Press.

Received February 2000; revised January 2001; accepted April 2001