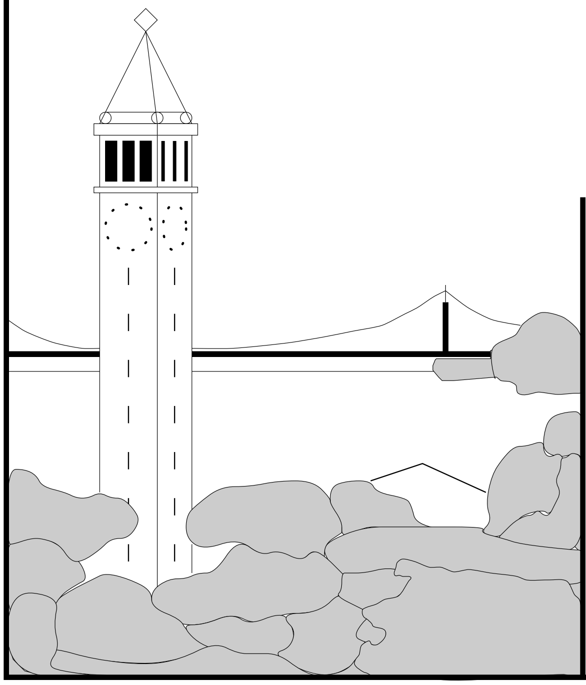
The State of the Art in Automated Usability Evaluation of User Interfaces

Melody Y. Ivory
Marti A. Hearst



Report No. UCB/CSD-00-1105

June 2000

Computer Science Division (EECS)
University of California
Berkeley, California 94720

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

Melody Y. Ivory
Computer Science Division
EECS Department
Berkeley, CA 94720-1776
ivory@cs.berkeley.edu

Marti A. Hearst
School of Information
Management and Systems
Berkeley, CA 94720-4660
hearst@sims.berkeley.edu

ABSTRACT

Usability evaluation is an increasingly important part of the iterative design process. Automated usability evaluation has great promise as a way to augment existing evaluation techniques, but is greatly underexplored. We present a new taxonomy for automated usability analysis and illustrate it with an extensive survey of evaluation methods. We present analyses of existing techniques, and suggest which areas of automated usability evaluation are most promising for future research.

Keywords

Automated Usability Evaluation, Performance Evaluation, User Interfaces, Web Interfaces, Taxonomy

Introduction

Usability is the extent to which a computer system can be used by users to achieve specified goals with effectiveness, efficiency and satisfaction in a given context of use.[1] Usability evaluation (UE) is a methodology for measuring these usability aspects of a system's user interface and identifying specific problems with the interface [23, 64]. Usability evaluation is an important part of the overall user interface iterative design process, which consists of cycles of designing, prototyping and evaluation [23, 64]. Usability evaluation is itself a process that entails many activities: specifying evaluation goals, identifying target users, selecting usability metrics, selecting an evaluation method and tasks, designing experiments, collecting usability data, and analyzing and interpreting data.

A wide range of usability evaluation techniques have been proposed, and a subset of these are currently in common use. Some evaluation techniques, such as formal user testing, can only be applied after the interface design 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 [44, 62, 63, 64]. Two studies in particular, the first and second comparative user testing studies (CUE-1 [62] and CUE-2 [63]), demonstrated less than a 1% overlap in findings among 4 and 8 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 [23, 64].

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 increase the number of study participants. An alternative is to make use of *automated* usability evaluation (AUE) methods.

Automated usability evaluation has several potential advantages over non-automated methods, including uncovering various types of errors more consistently, increasing the coverage of evaluated features, enabling comparisons between alternative designs, and predicting time and error costs across an entire design. They should reduce the need for evaluation expertise among individual developers and reduce the cost of usability evaluation as compared to standard techniques. Some automated evaluation techniques can be embedded within the design phase of UI development, as opposed to being applied after implementation. This is important because evaluation with most traditional methods can be done only after the interface has been built and changes are more costly [64].

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

Despite the potential advantages, the space of auto-

[1] From ISO9241 (*Ergonomic requirements for office work with visual display terminals* [41]).

mated usability evaluation is quite underexplored. In this article, we discuss the state of the art in automated usability evaluation, and highlight the approaches that merit further investigation. Section presents a taxonomy for classifying UE automation, and Section summarizes the application of this taxonomy to 128 usability methods. Sections – describe these methods in more detail, including our summative assessments of the methods. The results of this survey suggest promising ways to expand existing methods to better support automated usability evaluation.

Taxonomy of Automated Usability Evaluation

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 differ and in part because the usability methods discussed have often only been applied to one type or the other. 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 [38] and Human Factors Engineering [40] 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., user testing) techniques [19]. Whitefield, Wilson, and Dowell [101] present another classification scheme based on the presence or absence of a user and a computer. Neither of these taxonomies reflect the automation aspects of UE methods.

The sole existing survey of *automated* usability evaluation, by Balbo [6], uses a taxonomy which distinguishes among four features of automation:

- **Non Automatic** - no level of automation supported (i.e., evaluator performs method).
- **Automatic Capture** - software automatically captures interface usage (e.g., logging).
- **Automatic Analysis** - automatic identification of usability problems.
- **Automatic Critique** - automatic analysis coupled with automated suggestions for 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 user 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 critique method may require the evaluator to create a complex UI model as input. Thus, this classification scheme is somewhat misleading since it ignores the non-automated requirements of the UE methods.

We expand this taxonomy to include consideration of a method's non-automated testing requirements, both in terms of users and evaluators. We augment each of Balbo's features with an attribute called **testing level**; this indicates the human testing effort required for execution of the method:

- **Minimal Effort:** does not require testing or modeling.
- **Informal Use:** requires normal use (i.e., unstructured tasks completed by a user or evaluator).
- **Model Development:** requires the evaluator to develop a UI model and/or a user model in order to employ the method.
- **Formal Study:** requires a user or evaluator to complete a set of structured tasks and/or a procedure.

Finally, we group existing UE methods into the 5 general classes: testing, inquiry, inspection, analytical modeling and simulation.

- **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 to identify potential usability problems in an interface.
- **Inquiry:** users provide feedback on an interface via interviews, surveys, and other methods.
- **Analytical Modeling:** an evaluator employs user and interface models to generate quantitative 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).

Both testing and inspection are formative (i.e., they identify specific usability problems) unlike inquiry methods, which are summative (i.e., they provide general assessments of usability). Analytical modeling and simulation are engineering approaches to UE that enable evaluators to predict usability with user and interface models. Software engineering practices have had a major influence on the first three classes, while the latter two, analytical modeling and simulation, are quite similar to performance evaluation techniques used to analyze the performance of computer systems [42, 43]. Table 1 maps these method classes into automation and testing combinations for surveyed evaluation methods (see below).

In summary, the taxonomy consists of: a UE method type (testing, inquiry, inspection, analytical modeling and simulation); an automation type (none, capture, analysis and critique); and a testing level (minimal, informal, model and formal). In the remainder of this article, we use this taxonomy to analyze UE evaluation methods.

Summary of Automated Usability Evaluation Methods

We surveyed 70 UE methods applied to WIMP interfaces, and 58 methods applied to Web UIs. Of these 128 methods, only 27 apply to both Web and WIMP UIs. Table 2 combines survey results for both types of interfaces showing automation type and testing level. Table 2 contains only 101 methods, since we depict methods applicable to both UIs once. For some methods, we will discuss more than one approach; hence, we show the number of methods surveyed in parenthesis beside the testing level. There are major differences in automation among the 5 types of methods. Overall, automation patterns are similar for WIMP and Web interfaces, with the exception that analytical modeling and simulation methods are far less explored in the Web domain than for WIMP interfaces (2 vs. 17 methods). Appendix shows the information in Table 2 separated by UI type.

Table 2 shows that AUE in general is greatly underexplored. Non automatic methods represent 64% of the methods surveyed, while automated methods collectively represent only 36%. Of this 36%, automatic capture methods represent 17%, automatic analysis methods represent 17% and automatic critique methods represent 2%. All but two of the automatic capture and log file analysis methods require some level of testing; genetic algorithms and information scent modeling employ simulation to generate usage data. Hence, only 22% of the automated methods do not require formal testing or informal use to employ.

To be fully automated, an AUE method would provide the highest level of automation (i.e., critique) and require no user testing or informal use. Our survey found that this level of automation has been accomplished using only one method: guideline reviews (e.g., [27, 29, 57, 83]). Operationalized guidelines automatically detect and report usability violations and then make suggestions for fixing them (discussed further in Section).

Of those methods that support the next level of automation (i.e., analysis), Table 2 shows that analytical modeling and simulation methods represent the majority. All but one of these methods support automatic analysis without requiring formal testing or informal use. Most of these methods embed analysis within the design phase of UI development, as opposed to employment after development.

The next sections discuss the various UE types and their automation levels in more detail. Most methods are applicable to both WIMP and Web interfaces, however, we make distinctions where necessary about a method's applicability. We also present our assessments of automatic capture, analysis and critique techniques using the following criteria:

- **Effectiveness:** how well does a method inform UI improvements,
- **Ease of use:** how easy is a method to employ,
- **Effort to learn:** how easy is a method to learn, and
- **Applicability:** how widely applicable is a method to WIMP and/or Web UIs other than those originally applied to.

We discuss the effectiveness, ease of use, effort to learn, and applicability of automated methods within each class of techniques.

User Testing Methods

Usability testing with real participants is one of the most fundamental usability evaluation methods [64]. It provides an evaluator with direct information about how people use computers and what some of the problems are with the interface being tested. During usability testing, participants use the system or a prototype to complete a pre-determined set of tasks while the tester 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 user testing: automatic capture of use data and automatic analysis of this data according to some metrics or a model (referred to as log file analysis in Table

Automation Type	Testing Level: Minimal Effort	Informal Use	Model Development	Formal Study
Non Automatic		Inquiry		Testing Inspection Inquiry
Automatic Capture	Simulation	Inquiry Testing	Simulation	Testing Inspection Inquiry
Automatic Analysis	Inspection	Testing Simulation	Testing Analytical Modeling Simulation	Testing Simulation
Automatic Critique	Inspection		Inspection	

Table 1: Surveyed UE methods associated with combinations of automation type and testing level.

2). In rare cases methods support both automatically capturing and analyzing usage data [3, 96].

Automatic Capture Methods

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, a time-consuming activity. As an alternative, automatic 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.

Within the testing category of UE, automatic capture of usage data is supported by two methods: 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 [65].

Performance measurement techniques automatically record timestamps along with usage data, thus automatically and accurately aligning timing data with user interface events. Most video recording and event logging tools record timestamps along with usage data [3, 33, 96]. Some video recording tools (e.g., [33]) 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] [96] enables the evaluator to replay logged events, meaning it must be able to replicate logged events during playback. To replicate logged events, it needs the same study data (e.g., databases or documents) and expects the system to behave as it did during the study. Integrated Data Capture and Analysis Tool [33] 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) [3] supports event logging and screen capture via Java and so does not require special equipment. Both KALDI and UsAGE support log file analysis (see Section).

Remote testing is a method that enables testing between a evaluator and participant who are not co-located. In this case the evaluator is not able to observe the testing process directly, but can gather data about the process over a computer network. Remote testing methods are distinguished according to whether or not they are co-located in time or in location.

Same-time different-place and different-time different-place are two major remote testing approaches [34]. 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 during the test via a speaker telephone or the computer.

An example of a different-time different-place testing method is the journaled session [64], in which software guides the participant through a testing session and logs the results. Evaluators can use this approach with prototypes to get feedback early in the design process, as well as with released products. In the early stages, evaluators distribute disks containing a prototype of a software product and embedded code for recording users' actions. Users experiment with the prototype and return the disks to evaluators upon completion. It is also possible to embed dialog boxes within the prototype in order to record user comments or observations during usage. For released products, evaluators use this

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

UE Method	Automation Type				Description
	None	Capture	Analysis	Critique	
Testing (Formative)					
Thinking-aloud Protocol	F (1)				user talks during test
Question-asking Protocol	F (1)				tester asks user questions
Shadowing Method	F (1)				expert explains user actions
Coaching Method	F (1)				user can ask an expert questions
Teaching Method	F (1)				user teaches novice
Co-discovery Learning	F (1)				two users collaborate
Performance Measurement	F (1)	F (7)			capture usage and quantitative data
Log File Analysis			FIM (19)*		analyze captured usage data
Retrospective Testing	F (1)				review videotape with user
Remote Testing		FI (3)			distance testing
Inspection (Formative)					
Guideline Reviews	F (6)		(6)	M (13)†	guideline conformance
Cognitive Walkthrough	F (1)	F (1)			simulate problem solving
Pluralistic Walkthrough	F (1)				group cog. walkthrough
Heuristic Evaluation	F (1)				identify heuristic violations
Perspective-based Inspection	F (1)				narrowly focused heur. eval.
Feature Inspection	F (1)				evaluate product features
Formal Usability Inspection	F (1)				formal heur. eval.
Consistency Inspection	F (1)				UI consist. across products
Standards Inspection	F (1)				industry standard compliance
Inquiry (Summative)					
Contextual Inquiry	F (1)				field interviewing
Field Observation	F (1)				observe system use
Focus Groups	F (1)				user group discussion
Interviews	F (1)				formally ask user questions
Surveys	F (1)	I (1)			informal interview
Questionnaires	F (1)	I (1)			subjective evaluation
Self-reporting Logs	FI (1)				user records UI operations
Screen Snapshots	FI (1)				user captures UI screens
User Feedback	F (1)				user-initiated comments
Analytical Modeling (Predictive)					
GOMS Analysis			M (4)		execution & learning time
UIDE Analysis			M (2)		analysis within a UIDE
Programmable User Models			M (1)		programming UI to fit user
Simulation (Predictive)					
Information Processor Model			M (9)		simulating user interaction
Petri Nets			FM (1)		simulating user interaction
Genetic Algorithms		(1)			simulating novice user interaction
Information Scent Model		M (1)			simulating Web site navigation
Automation Type					
Total	26	7	7	1	
Percent	64%	17%	17%	2%	

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

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. This can be especially problematic for same-time different-place 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) [34]. 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.

The Web enables remote testing and performance measurement on a much larger scale than is economically feasible with WIMP interfaces. Similar to journaled sessions, Web servers maintain usage logs and automatically generate a log file entry for each request. These entries include the IP address of the requester, request time, name of the requested Web page, and in some cases the URL of the referring page (i.e., where the user came from). Server logs track only *unique* navigational events, since they cannot record user interactions that occur on the client side only (e.g., use of within-page anchor links, back button or cached pages). Furthermore, the validity of the data is questionable due to caching by proxy servers and browsers [25, 83]. Server logs may not reflect usability, especially since these logs are often difficult to interpret [84].

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, however, 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 [83] supports remote testing of a Web site. This suite includes WebVIP (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. (There is also a visualization tool, VISVIP [20], for viewing logs collected with WebVIP; see Section .) Using this client-side data, the evaluator can accurately measure time spent on tasks or particular pages as well as study use of the back button and user paths. 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 difficulties getting the copied site to function properly. The evaluator must also add logging code to each individual link on a page. Since WebVIP only collects data on selected HTML links, it does not record interactions with other Web objects, such as forms. It also does not record usage of external or non-instrumented links.

Similar to WebVIP, the Web Event-logging Tool (WET) [25] supports the capture of client-side data, including clicks on Web objects, window resizing, typing in a form object and form resetting. WET interacts with Microsoft Internet Explorer and Netscape Navigator to record browser event information, including the type of event, a time stamp, and the document-window location. This gives the evaluator a more complete view of the user's interaction with a Web interface than WebVIP. WET does not require as much effort to employ as WebVIP, nor does it suffer from the same limitations. To use this tool, the evaluator specifies events (e.g., clicks, changes, loads and mouseovers) and event handling functions in a text file on the Web server; sample files are available to simplify this step. The evaluator must also add a single call to the text file within the HEAD tag of each Web page to be logged. Currently, the log file analysis for both WebVIP and WET is manual. Future work has been proposed to automate this analysis.

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 [64]. In non-automated card sorting, the evaluator (or a team of evaluators) writes concepts on pieces of paper, and users group the topics into piles. The evaluator manually analyzes these groupings to determine a good category structure. WebCAT allows the evaluator to test proposed topic categories for a site via a category matching task; 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, and helps make the technique scale for a large number of topic categories.

Automatic 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.

It is difficult to assess the ease of use and learning of these approaches, especially the Integrated Data Capture and Analysis Tool 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, while WebVIP requires the evaluator to add code to every link on all Web pages. WET also enables the evaluator to capture more comprehensive usage data than WebVIP. 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.

Automatic Analysis Methods

Log file analysis methods support automatic analysis of data captured during formal testing or informal use. Since Web servers automatically log client requests, log file analysis is a heavily used methodology for evaluating Web interfaces [24, 30, 36, 92]. Our survey reveals five general approaches for analyzing WIMP and Web log files: metric-based, pattern-matching, task-based, task-based pattern-matching, and inferential.

Metric-based Analysis of Log Files. Metric-based approaches generate quantitative performance measurements. Three examples for WIMP interfaces are DRUM [60], the MIKE UIMS [66], and AMME (Automatic Mental Model Evaluator) [76, 78, 79]. DRUM enables the evaluator to review a video tape of a user test and manually log starting and ending points for tasks. DRUM processes this log and derives the following measurements: task effectiveness (i.e., how correctly and completely tasks are completed), user efficiency (i.e., effectiveness divided by task completion time), productive period (i.e., portion of time the user did not have problems) and learnability (i.e., comparison of the user's and expert user's efficiency for a task). 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 function 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 [72] to reconstruct the user's model (i.e., problem solving process) and analyzes this model. 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 net. It then computes measures of behavioral complexity (i.e., steps taken to perform tasks), routinization (i.e., repetitive use of task sequences), and ratios of thinking vs. waiting time. User studies with novices and experts validated these quantitative measures and showed behavioral complexity to correlate negatively with learning (i.e., less steps are taken to solve tasks as a user learns the interface) [79]. Hence, this measure provides insight on interface complexity. It is also possible to simulate the generated petri net (see Section) to further analyze the user's task 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 [86] and others [5] allow evaluators to pinpoint performance bottlenecks, such as slow server response time, that may negatively impact the usability of a Web site. Service Metrics' tools supports this kind of performance analysis, including software that can collect these measures from multiple geographical locations under various access conditions. These approaches focus on server and network performance, but provide little insight into the usability of the Web site itself.

Pattern-Matching Analysis of Log Files. Pattern-matching approaches, such as MRP (Maximum Repeating Pattern) [87], 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.

Task-based Analysis of Log Files. Task-based approaches analyze discrepancies between the designer and

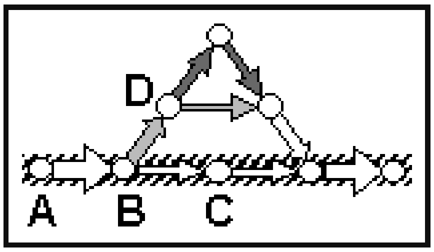


Figure 1: QUIP usage profile contrasting task flows for two users to the designer's task flow (diagonal shading). Each node represents a user action, and directed arcs indicate actions taken by users. The width of arcs denote the fraction of users completing actions, while the color of arcs reflect the average time between actions (darker colors correspond to longer time).

user task models. For example, the IBOT system [107] 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 higher-level abstractions (e.g., menu select and menubar search operations). Evaluators can use the system to compare user and designer behavior on high-level 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 support automated critique.

The QUIP (Quantitative User Interface Profiling) tool [35] and KALDI provide the most 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-based and trace-based information into directed graphs (see Figure 1). 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 1. Currently, the evaluator must instrument the UI to collect the necessary usage data, and must manually analyze the graphs to identify usability problems.

KALDI [3] automatically captures usage data and screen shots for Java applications (see previous section). It also enables the evaluator to classify tasks (both manually and via automatic filters), compare user performance on tasks, and playback synchronized screen shots. It depicts logs graphically in order to facilitate analysis. UsAGE, which also supports automatic logging within a UIMS, provides a similar graphical presentation for comparing event logs for expert and novice users. 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.

Task-based Pattern-matching Analysis of Log Files. ÉMA (Automatic Analysis Mechanism for the Ergonomic Evaluation of User Interfaces) [7] and USINE (USer Interface Evaluator) [53] combine task-based and pattern-matching techniques.

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

USINE (USer Interface Evaluator) [53] employs the ConcurTaskTrees [70] 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 (Remote USer Interface Evaluator) [71] is an extension that analyzes multiple log files (typically captured remotely) to enable comparison across users.

Inferential Analysis of Log Files. Inferential analysis of Web log files includes both statistical and visualization techniques. Statistical approaches include traffic-based analysis (e.g., pages-per-visitor or visitors-per-page) and time-based analysis (e.g., click paths and page-view durations) [24, 30, 92, 93]. Some methods require manual pre-processing or filtering of the logs before analysis. Furthermore, the evaluator must interpret reported measures in order to identify usability problems. This 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 [12, 84]. Nonetheless, inferential analysis techniques have been useful for improving usability and enable ongoing, cost-effective evaluation throughout the life of a site [30, 92].

Visualization is also used for inferential analysis of Web and WIMP log files [32, 36]. Starfield visualization [36] is one approach that enables evaluators to interactively explore 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 data points (e.g., URLs requested versus time of requests) in an interface that supports zooming, filtering and dynamic querying [2]. 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. As such, it would be beneficial to employ a statistical inferential approach, such as time-based log file analysis, prior to exploring visualizations.

Dome Tree visualization [15] provides a more insightful representation of simulated (see Section) and Web usage captured in server log files. This approach maps a Web site into a three dimensional surface representing the hyperlinks (see top part of Figure 2). The location of links on the surface are determined by a combination of content similarity, link usage and link structure of Web pages. The visualization highlights the most commonly traversed subpaths. An evaluator can explore these usage paths to possibly gain insight about the information "scent" (i.e., common topics among Web pages on the path) as depicted in the bottom window of Figure 2. 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 employ a statistical inferential approach prior to site exploration with this approach.

VISVIP [20] is a three-dimensional tool for visualizing log files compiled by WebVIP during user testing (see previous section). Figure 3 shows VISVIP's Web site (top graph) and usage path (bottom graph) depictions to be similar to the Dome Tree visualization approach. VISVIP generates a 2D layout of the site using a force-directed algorithm, wherein adjacent nodes are placed closer together than non-adjacent nodes. The third dimension reflects path 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 task completion, prior statistical inferential processing is not necessary for VISVIP usage.

Although the log file analysis techniques vary widely on the four assessment criteria, all approaches offer substantial benefits over the alternative – time-consuming, unaided analysis of potentially large amounts of raw data. Task-based and 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 simpler pattern-matching approaches; this additional effort is mainly in the development of task models. Although pattern-matching approaches are easier to use and learn, they only detect problems for pre-specified usage patterns.

Metric-based approaches in the WIMP domain have been effective at associating measurements with specific interface aspects (e.g., commands and tasks), which can then be used to identify usability problems. AMME also helps the evaluator to understand the user's model of an interface and conduct simulation studies. However, metric-based approaches require the evaluator to conduct more analysis 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 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.

Inspection Methods

A usability inspection is an informal evaluation methodology whereby an evaluator examines the usability aspects of a UI design with respect to its conformance to a set of guidelines. Unlike other UE methods, inspections rely solely on the evaluator's judgment as a source of evaluation feedback. A large number of detailed usability guidelines have been developed for WIMP [67, 89] and Web [17, 22, 55, 59, 99] interface design. Common non-automated inspection techniques are heuristic evaluation [64] and cognitive walkthroughs [56].

Designers have historically experienced difficulties following design guidelines [10, 21, 57, 88]. One study has also demonstrated that designers are biased towards

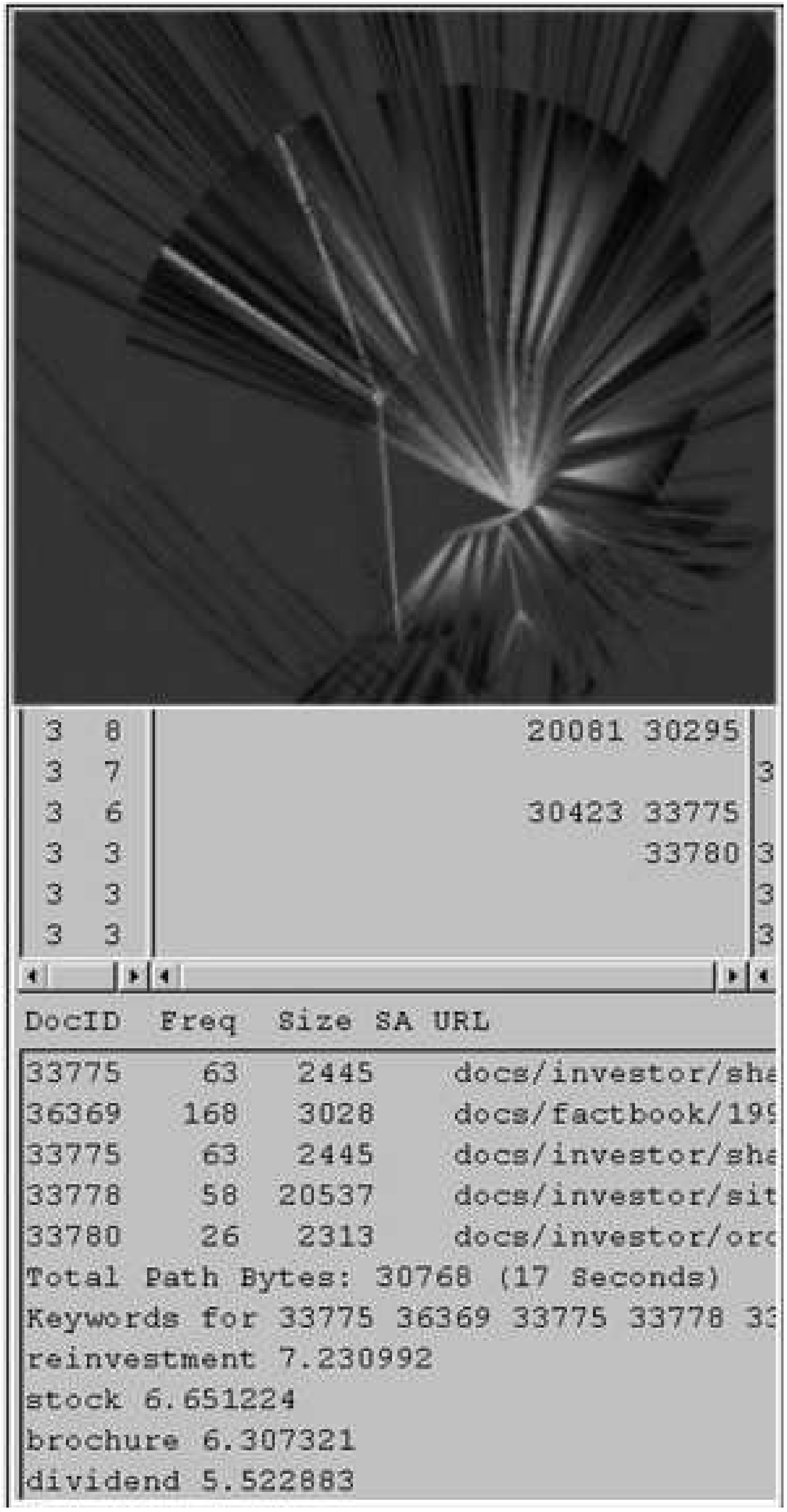


Figure 2: Dome Tree visualization of a Web site with a usage path displayed. The bottom part of the figure displays information about the usage path, including an estimated navigation time and information scent (i.e., common keywords along the path).

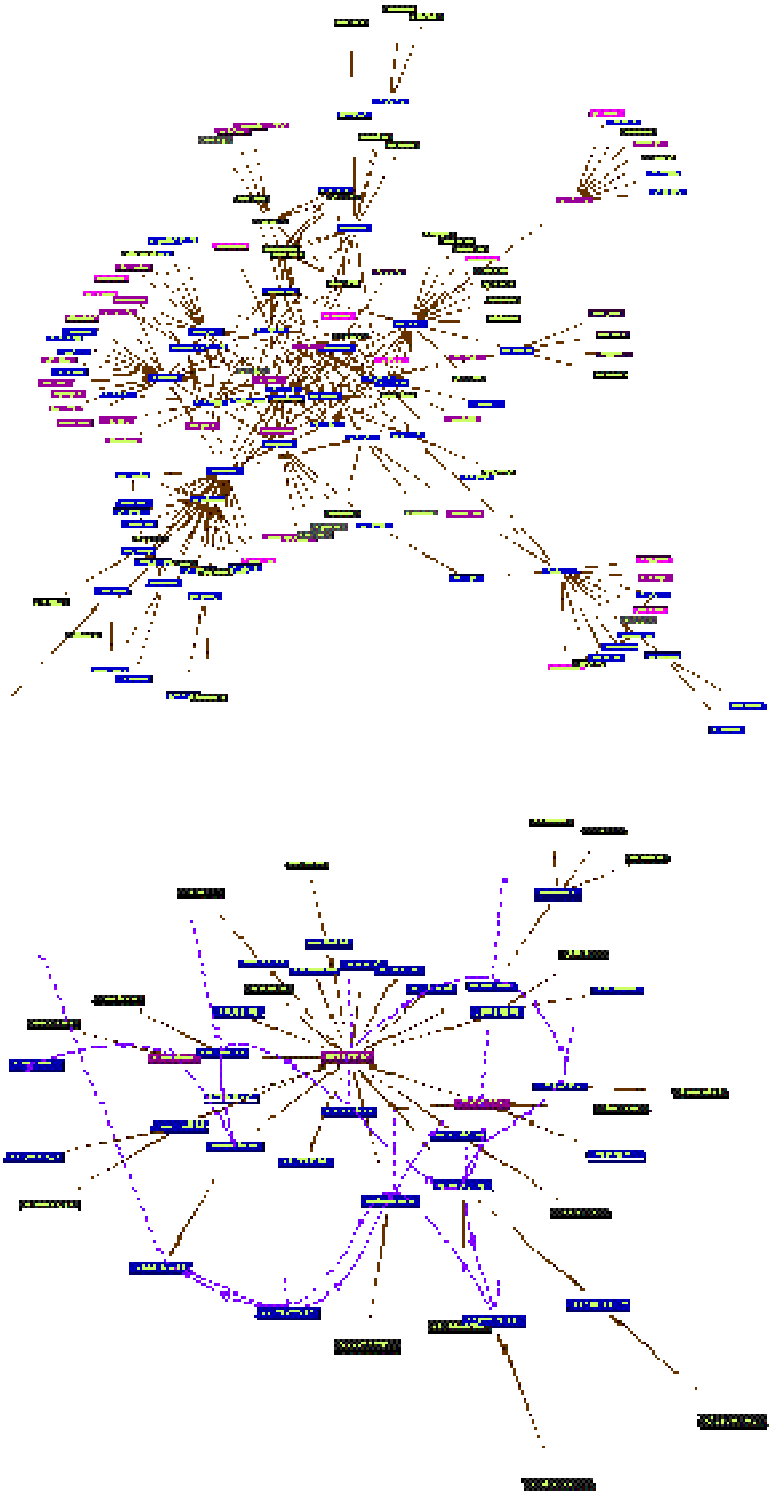


Figure 3: VISVIP visualization of a Web site. The bottom part of the figure displays a usage path laid over the site.

aesthetically pleasing interfaces, regardless of efficiency [85]. Screen layout tools, especially validated ones, assist designers with objective evaluation of WIMP UIs. Such tools have been effective at identifying visual problems (e.g., inefficient screen usage, misaligned elements, or size imbalance among elements), but they cannot detect logical and semantic problems that arise during usage. Although such tools appear easy to use and learn, their application is dependent on the development platform employed.

Because designers have difficulty applying design guidelines, automation has been predominately used within the inspection class to check guideline conformance. One notable method is operationalized guidelines, which automatically detect and report usability violations and in some cases make suggestions for fixing them. Automated capture, analysis, and critique methods have been applied to other inspection methods, as described in the following subsections.

Automatic Capture Methods

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 [81]. Unfortunately, evaluators found this approach too cumbersome and time-consuming to employ.

Automatic Analysis Methods

Several quantitative measures have been proposed for evaluating interfaces. Tullis [95] derived size measures (Overall Density, Local Density, Number of Groups, Size of Groups, Number of Items, and Layout Complexity). Streveler and Wasserman [91] proposed "boxing," "hotspot," and "alignment" analysis techniques. These early techniques were designed for alphanumeric displays, while more recent techniques evaluate WIMP interfaces. Vanderdonckt and Gillo [98] 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 [77] proposed and validated four measures (Functional Feedback, Interactive Directness, Application Flexibility, and Dialog Flexibility) to evaluate WIMP UIs. Quantitative measures have been incorporated into automatic layout tools [9, 51] as well as several automatic analysis tools [61, 69, 85], discussed immediately below.

Parush, Nadir, and Shtub [69] developed and validated a tool for computing the complexity of Visual Basic dialog boxes. 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 (semi-Automated Interface Designer and Evaluator) [85] 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 [61] is another automatic analysis tool for Windows interfaces. Rather than assessing ergonomic factors, it focuses on task-independent consistency checking (e.g., same widget placement and labels) within the UI or across multiple UIs; user studies have shown a 10-25% speedup for consistent interfaces [61]. 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 Visual Basic and Visual C++ resource files.

The Rating Game [90] 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 [93] and Gentler [94], 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. (HyperAT also supports inferential analysis of server log files similar to other log file analysis techniques; see Section .) Gentler [94] provides similar structural analysis but focuses on maintenance of existing sites rather than design of new ones.

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 [52, 106], general thresholds still remain to be established. On the other hand, these approaches are easy to use, learn and apply to all Web UIs.

Automatic Critique Methods

Critiques give designers clear directions for conforming to violated guidelines and consequently improving usability. As mentioned above, following guidelines has historically been problematic, especially for a large number of guidelines. Automatic critique approaches, especially ones that modify a UI, provide the highest level of support for adhering to guidelines.

The KRI/AG tool (Knowledge-based Review of user Interface) [57] is an automatic critique that checks the guideline conformance of X Window UI designs created using the TeleUSE UIMS [54]. KRI/AG contains a knowledge base of guidelines and style guides, including the Smith and Mosier guidelines [89] and Motif style guides [67]. It uses this information to automatically critique a UI design and generate comments about possible flaws in the design. IDA (user Interface Design Assistance) [80] also embeds rule-based (i.e., expert system) guideline checks within a UIMS. SYNOP [6] is a similar automatic 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) [45] is another approach that 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 [29] 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 [68] to enable guideline checks throughout the development process.

All of these 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. For example, Farenc, Liberati, and Barthet [28] 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 is that they require considerable modeling and learning effort on the part of the evaluator. All methods, except Ergoval, also suffer from limited applicability.

Several automatic critique tools use guidelines for Web site usability checks (see [103] for a listing). The World Wide Web Consortium's HTML Validation Service [102] checks that HTML code conforms to standards. Weblint [11] and Dr Watson [1] also check HTML syntax and in addition verify links. Dr Watson also computes download speed and spell checks text.

The Web Static Analyzer Tool (SAT) [83], part of the NIST WebMetrics suite of tools, assesses static HTML according to a number of usability guidelines, such as whether all graphics contain ALT tags, the average number of words in link text, and the existence of at least one outgoing link on a page. 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. UsableNet's LIFT Online and LIFT Onsite [97] perform similar usability checks 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 Onsite guides users through making suggested improvements. Bobby [16, 18] is another HTML analysis tool that checks Web pages for their accessibility [99] 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. However, Ratner, Grose, and Forsythe [75] question the validity of HTML usability guidelines, since most have not been subjected to a rigorous development process as established guidelines for WIMP interfaces. Analysis of 21 HTML guidelines showed little consistency among them, with 75% of recommendations appearing in only one style guide. Furthermore, only 20% of HTML-relevant recommendations from established guidelines existed in the 21 HTML style guides. WebEval [82] 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 Ergoval, some problems, such as whether text will be understood by users, are difficult to detect automatically.

Unlike other automatic critique approaches, the Design Advisor [27] enables visual critique of Web pages. The tool uses empirical results from eye tracking studies de-

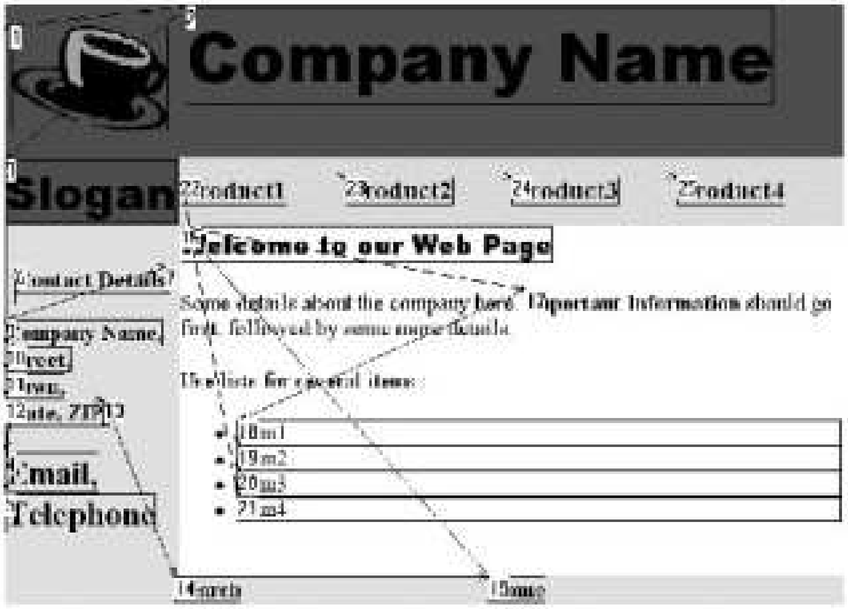


Figure 4: The Design Advisor superimposes a scanning path on the Web page. The numbers indicate the order in which elements will be scanned.

signed to assess the attentional effects of various elements, such as animation, images, and highlighting, in multimedia presentations [26]; 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 4. It also provides suggestions for improving scanning paths.

Inquiry Methods

Similar to user testing approaches, inquiry methods require feedback from users and are often employed during user testing. The focus, however, 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 usually employ inquiry methods, such as surveys, questionnaires, and interviews, to gather supplementary data after a system is released.[3] Inquiry methods are summative in nature and provide feedback on the overall quality of the interface, such as whether users like it, generally have problems with it, and so on. This is useful information for improving the interface for future releases. These methods vary based on whether the evaluator interacts with a user or a group of users or whether users report their experiences using questionnaires, surveys or usage logs possibly in conjunction with screen snapshots.

Automation has been used predominately for automatically capturing use data during formal testing or informal use. Interactive surveys and questionnaires can be embedded into a user interface to semi-automate the usage capture process. The Web inherently facilitates automatic capture of survey and questionnaire data using forms. These approaches enable the evaluator to collect subjective usability data and possibly make improvements throughout the life of an interface.

As previously discussed, automatic capture methods represent an important first step toward informing UI improvements. Automated inquiry methods make it possible to collect data quickly from a larger number of users than is typically possible with non-automated methods. However, automated inquiry methods suffer from the same limitation of non-automated approaches – they may not clearly indicate usability problems due to the subjective nature of user responses. Furthermore, they do not enable automated analysis or critique of interfaces. The real value of these techniques is that they are easy to use and widely applicable.

Analytical Modeling Methods

Analytical modeling complements traditional evaluation techniques like user testing. Given some representation or model of the UI and/or user, these methods inexpensively generate quantitative usability predictions. Automation has been predominately used to analyze task completion (e.g., execution and learning time) within WIMP UIs and Web site structure (e.g., breadth and depth). Analytical modeling inherently supports automatic analysis. Our survey did not reveal analytical modeling techniques to support automated critique. Most analytical modeling and simulation approaches for

[3] Contextual inquiry [37] is an exception to this; it is a needs assessment method used early in the design process.

WIMP and Web UIs are based on the model human processor (MHP) proposed by Card et al. [14]. GOMS analysis is one of the most widely accepted analytical modeling methods based on the MHP [46]. Other methods based on the MHP employ simulation and will be discussed in the next section.

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 predict task execution and learning times for error-free expert performance. The four approaches in this family include the original GOMS method proposed by Card, Moran and Newell (CMN-GOMS) [14], the simpler keystroke-level model (KLM), the natural GOMS language (NGOMSL) and the critical path method (CPM-GOMS) [46]. These approaches differ in the task granularity modeled (e.g., keystrokes or a high-level procedure) and in the support for alternative methods (i.e., selections) and multiple goals.

Two of the major roadblocks to using GOMS have been the tedious task analysis and the need to estimate execution and learning times. These must be specified and computed manually. USAGE[4] (the UIDE System for semi-Automated GOMS Evaluation) [13] and CRITIQUE (the Convenient, Rapid, Interactive Tool for Integrating Quick Usability Evaluations) [39] are tools that address these limitations by automatically generating a 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) [49] is another tool that generates quantitative predictions for a given GOMS task model (discussed in more detail in Section). 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 their wide application is that they operate on limited platforms (e.g., Sun machines), model low-level goals (e.g., at the keystroke level for CRITIQUE), and do not support multiple ways of accomplishing tasks because they use an idealized expert user model.

The Programmable User Model (PUM) [104] is an entirely different analytical modeling technique for automatic analysis. In this approach, the designer is required to write a program that acts like a user using the interface design; the designer must specify explicit sequences of operations for each task. These are executed within an architecture that imposes approximations of psychological constraints, such as memory limitations. Difficulties experienced by the designer while programming the architecture can then be used to improve the UI. Once the designer successfully programs the architecture, the model can be executed 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.

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 UIs and is effective at predicting usability problems. However, these predictions are limited to error-free expert performance. 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. PUMs, however, still requires considerable effort and learning time to employ, since it is a programming approach. 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 like GOMS analysis will not map as well to the Web domain, because it is difficult to predict how a user will accomplish the goals in a task hierarchy given that there are many different ways to navigate a typical site. Another problem is GOMS' reliance on an expert user model, which does not fit the diverse user community of the Web. Hence, new analytical modeling approaches are required to evaluate the usability of Web sites.

Simulation Methods

Simulation complements traditional UE methods and, like analytical modeling, inherently supports automatic 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. Simulation is also used to automatically generate usage data for analysis with log file analysis techniques [15] or event playback in a UI [47]. Hence, simulation also supports automatic capture. Evaluators can run simulations with different parameters in order to study various UI design tradeoffs and thus make more informed decisions about UI implementation.

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

Automatic Capture

Kasik and George [47] developed an automatic capture technique for driving tools that replay events (such as executing a log file) for 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) that an evaluator can then use to find weak spots, failures, and other usability problems during the design phase. The system enables a designer to generate an expert user trace and then insert deviation commands at different points within the trace. It uses a genetic algorithm to determine user behavior during 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 evaluator to simulate other user models. Although currently not supported by this tool, traditional random number generation can also be employed to explore the outer limits of a UI (i.e., completely random user behavior).

Without such an automated capture technique, the evaluator must anticipate all possible usage scenarios or rely on user testing or informal use to generate usage traces. Testing and informal use limit UI coverage to a small number of tasks or to UI features that are employed in regular use. Automated capture 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. This system appears to be relatively straightforward to use, since it interacts directly with a running application and does not require modeling. Interaction with the running application also ensures that generated usage traces are plausible. Experiments demonstrated that it is possible to generate a large number of usage traces within an hour. However, an evaluator must manually analyze the execution of each trace in order to identify problems. The authors propose future work to automatically verify that a trace produced the correct result. Currently, this tool is only applicable to Motif-based UIs.

Chi, Pirolli, and Pitkow [15] developed a similar automatic capture approach for generating navigation paths for Web UIs. This approach creates a model of an existing site that embeds information about the similarity of content among pages, captured usage data, and linking structure. The evaluator specifies starting points in the site and information needs (i.e., target pages) as input to the simulator. The 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 target pages or after an arbitrary amount of effort (e.g., maximum number of links or browsing time). The simulator records navigation paths and reports the proportion of agents that reached target pages.

The authors use these usage paths as input to the Dome Tree visualization methodology, an inferential log file analysis approach discussed in Section . The authors compared actual and simulated navigation paths for Xerox's corporate site and discovered a close match when scent is clearly visible (i.e., not buried under graphics or text). 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.

Automatic Analysis

AMME [79] (see Section) 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, 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, which has been shown to correlate negatively with learning and interface complexity. Studies have also validated the accuracy of generated models with usage data [76]. AMME should be applicable to Web interfaces as well, since it constructs models from log files. Despite its advantages, AMME still requires formal user testing to generate log files for simulation studies.

The remaining WIMP simulations rely on some variation of a human information processor model similar to the MHP previously discussed. Pew and Mavor [73] provide a detailed discussion of this type of modeling and an overview of many of these approaches, including five that we discuss: ACT-R (Adaptive Control of Thought) [4], COGNET (COGnition as a NEtwork of Tasks) [105], EPIC (Executive-Process Interactive Control) [50], HOS (Human Operator Simulator) [31] and Soar [74]. Here, we also consider CCT (Cognitive Complexity Theory) [48], ICS (Interacting Cognitive Subsystems) [8] and GLEAN (GOMS Language Evaluation and ANalysis) [49]. Rather than describe each method individually, we summarize the major characteristics of these simulation methods in Table 3 and discuss them below.

Modeled Tasks The models we surveyed simulate the following 3 types of tasks: a user performing cognitive tasks (e.g., problem-solving and learning: COGNET, ACT-R, Soar, ICS); a user immersed in a human-machine system (e.g., an aircraft or tank: HOS); and a user interacting with a typical UI (EPIC, GLEAN, CCT).

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

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

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

Predictions The surveyed methods return a number of simulation results, including predictions of task performance (EPIC, CCT, COGNET, GLEAN, HOS, Soar), memory load (ICS, CCT), learning (ACT-R, SOAR, ICS, GLEAN, CCT), or behavior predictions such as action traces (ACT-R, COGNET, EPIC).

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

EPIC is the only simulation analysis method that embodies most of these ideal characteristics. It 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 and all of the other methods require considerable learning time and effort to employ. They are also applicable to a wide range of UIs.

Our survey revealed only one simulation approach for automatic analysis of Web interfaces – WebCriteria's Site Profile [100]. Unlike the other simulation approaches, it requires an implemented interface for evaluation. Site Profile performs analysis in four phases: gather, model, analyze and report. During the gather phase, a spider traverses a site (200-600 unique pages) to collect Web site data. This data is then used to construct a nodes-and-links model of the site. For the analysis phase, it uses a standard Web user model (called Max [58]) 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. This method 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.

Expanding Existing Automated Usability Evaluation Methods

Automated usability evaluation has many potential benefits, including reducing the costs of non-automated methods, aiding in comparisons between alternative designs and for improving consistency in problems found. Research to further develop analytical modeling, simulation and log file analysis techniques could result in several promising AUE techniques as discussed below.

Our survey showed log file analysis to be a viable methodology for automated analysis of usage data. However, it still requires formal testing or informal use to employ. One way to expand the use and benefits of this

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

Table 3: Characteristics of simulation methods surveyed.

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, genetic algorithms and information scent modeling, that automatically generate plausible usage data. Genetic algorithms determine user behavior during deviation points in an expert user script, while the information scent model selects navigation paths by considering information scent. Hence, both of these approaches generate plausible usage traces without user testing or informal use. These techniques also provide valuable insight on leveraging 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 as well.

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). After mapping task sequences into specific UI operations in each interface, the benchmark could be executed within each UI to collect measurements. This is a promising open area of research for evaluating comparable WIMP UIs.

Given a wider sampling of usage data, task-based pattern-matching 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, for understanding user behavior, preferences and errors. Although the authors claim that this approach works well for WIMP UIs, it needs to be adapted to work for Web UIs where tasks may not be clearly-defined. Additionally, since USINE already reports substantial analysis data, this data could be compared to usability guidelines in order to support automated critique.

Our survey also showed that evaluation within a user interface development environment (UIDE) is a promising approach for automated analysis. The AIDE approach provides the most support for evaluating and improving UI designs and could be expanded to Web interfaces. Guidelines could also be incorporated into AIDE analysis to support automatic critique. 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. Applying such analysis approaches outside of these user interface development environments is an open research problem.

In addition, 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 AUE, especially for evaluating alternative designs.

Several simulation techniques employed in the performance analysis of computer systems, in particular traced-driven discrete-event simulation and Monte Carlo simulation [43], would enable designers to perform what-if analysis 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. AMME, however, 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., a probability distribution over possible events determines what event occurs next). Monte Carlo simulation could contribute substantially to automated UE by eliminating the need for explicit task hierarchies or user models. Most simulations in this domain rely on a single user model, typically an expert user. This would enable designers to perform what-if analysis and study design alternatives with many user models. The approach employed by Chi, Pirolli, and Pitkow [15] to simulate Web site navigation is a close approximation to Monte Carlo simulation.

Conclusions

In this article we provided an overview of automated usability evaluation and presented a taxonomy for comparing various methods. We also presented an extensive survey of AUE methods for WIMP and Web interfaces, finding that AUE methods represent only 36% of methods surveyed. Of these methods, only 22% are free from requirements of formal testing or informal use. Of these, all approaches, with the exception of operationalized guidelines, are based on analytical modeling or simulation.

It is important to keep in mind that AUE does not capture important qualitative and subjective information (such as user preferences and misconceptions) that can only be unveiled via user 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 automated approaches in tandem with non-automated methods, such as heuristic evaluation and user testing. For example, an evaluator doing a heuristic evaluation could observe automatically-generated usage traces executing within a UI.

Automated usability evaluation has many potential benefits, including reducing the costs of non-automated methods, aiding in comparisons between alternative designs and for improving consistency in problems found. Research to further develop analytical modeling, simulation and log file analysis techniques could result in several promising AUE techniques.

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 2.

Acknowledgments

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

1. Addy & Associates. Dr. watson version 4.0, 2000. Available at http://watson.addy.com/.

2. Christopher Ahlberg and Ben Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Human Factors in Computing Systems. Conference Proceedings CHI'94*, pages 313–317, 1994.

3. Ghassan Al-Qaimari and Darren McRostie. KALDI: A computer-aided usability engineering tool for supporting testing and analysis of human-computer interaction. In J. Vanderdonckt and A. Puerta, editors, *Proceedings of the 3rd International Conference on Computer-Aided Design of User Interfaces (CADUI'99)*, Dordrecht, October 1999. Louvain-la-Neuve, Kluwer.

4. John R. Anderson. *The Adaptive Character of Thought*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

5. Beth Bacheldor. Push for performance. *Information Week*, September 20:18–20, 1999.

6. Sandrine Balbo. Automatic evaluation of user interface usability: Dream or reality. In *Proceedings of QCHI 95*, 1995.

7. Sandrine Balbo. ÉMA: Automatic analysis mechanism for the ergonomic evaluation of user interfaces. Technical Report 96/44, DSIRO Division of Informaiton Technology, 1996.

UE Method	None	Capture	Analysis	Critique	Description
	Automation Type				
Testing (Formative)					
Thinking-aloud Protocol	F (1)				user talks during test
Question-asking Protocol	F (1)				tester asks user questions
Shadowing Method	F (1)				expert explains user actions
Coaching Method	F (1)				user can ask an expert questions
Teaching Method	F (1)				user teaches novice
Co-discovery Learning	F (1)				two users collaborate
Performance Measurement	F (1)	F (3)			capture usage and quantitative data
Log File Analysis			FIM (10)*		analyze captured usage data
Retrospective Testing	F (1)				review videotape with user
Remote Testing		FI (2)			distance testing
Inspection (Formative)					
Guideline Reviews	F (2)		(3)	M (5)†	guideline conformance
Cognitive Walkthrough	F (1)	F (1)			simulate problem solving
Pluralistic Walkthrough	F (1)				group cog. walkthrough
Heuristic Evaluation	F (1)				identify heuristic violations
Perspective-based Inspection	F (1)				narrowly focused heur. eval.
Feature Inspection	F (1)				evaluate product features
Formal Usability Inspection	F (1)				formal heur. eval.
Consistency Inspection	F (1)				UI consist. across products
Standards Inspection	F (1)				industry standard compliance
Inquiry (Summative)					
Contextual Inquiry	F (1)				field interviewing
Field Observation	F (1)				observe system use
Focus Groups	F (1)				user group discussion
Interviews	F (1)				formally ask user questions
Surveys	F (1)	I (1)			informal interview
Questionnaires	F (1)	I (1)			subjective evaluation
Self-reporting Logs	FI (1)				user records UI operations
Screen Snapshots	FI (1)				user captures UI screens
User Feedback	F (1)				user-initiated comments
Analytical Modeling (Predictive)					
GOMS Analysis			M (4)		execution & learning time
UIDE Analysis			M (2)		analysis within a UIDE
Programmable User Models			M (1)		programming UI to fit user
Simulation (Predictive)					
Information Processor Model			M (8)		simulating user interaction
Petri Nets			FM (1)		simulating user interaction
Genetic Algorithms		(1)			simulating novice user interaction
Automation Type					
Total	26	6	7	1	
Percent	65%	15%	17%	3%	

Table 4: Automation characteristics of 70 WIMP UE methods. A number in parentheses indicates the number of methods surveyed for a particular method and automation type. The testing level for each method is represented as: minimal (blank), formal (F), informal (I) and model (M). The * for the FIM entry indicates that either formal or informal testing is required. In addition, a model may be used in the analysis. The † indicates that methods may or may not employ a model. Four methods support multiple levels of automation: DRUM - performance measurement and log file analysis; AMME - log file analysis and petri net simulation; KALDI - performance measurement, log file analysis, and remote testing; and UsAGE - performance measurement and log file analysis.

UE Method	Automation Type: None	Capture	Analysis	Critique	Description
Testing (Formative)					
Thinking-aloud Protocol	F (1)				user talks during test
Question-asking Protocol	F (1)				tester asks user questions
Shadowing Method	F (1)				expert explains user actions
Coaching Method	F (1)				user can ask an expert questions
Teaching Method	F (1)				user teaches novice
Co-discovery Learning	F (1)				two users collaborate
Performance Measurement	F (1)	F (4)			capture usage and quantitative data
Log File Analysis			FIM (9)		analyze captured usage data
Retrospective Testing	F (1)				review videotape with user
Remote Testing		FI (1)			distance testing
Inspection (Formative)					
Guideline Reviews	F (4)		(3)	(8)	guideline conformance
Cognitive Walkthrough	F (1)				simulate problem solving
Pluralistic Walkthrough	F (1)				group cog. walkthrough
Heuristic Evaluation	F (1)				identify heuristic violations
Perspective-based Inspection	F (1)				narrowly focused heur. eval.
Feature Inspection	F (1)				evaluate product features
Formal Usability Inspection	F (1)				formal heur. eval.
Consistency Inspection	F (1)				UI consist. across products
Standards Inspection	F (1)				industry standard compliance
Inquiry (Summative)					
Contextual Inquiry	F (1)				field interviewing
Field Observation	F (1)				observe system use
Focus Groups	F (1)				user group discussion
Interviews	F (1)				formally ask user questions
Surveys	F (1)	I (1)			informal interview
Questionnaires	F (1)	I (1)			subjective evaluation
Self-reporting Logs	FI (1)				user records UI operations
Screen Snapshots	FI (1)				user captures UI screens
User Feedback	F (1)				user-initiated comments
Analytical Modeling (Predictive)					
No Methods Surveyed					
Simulation (Predictive)					
Information Processor Model			M (1)		simulating user interaction
Information Scent Model		M (1)			simulating Web site navigation
Automation Type					
Total	26	5	3	1	
Percent	74%	14%	9%	3%	

Table 5: Automation characteristics of 58 Web UE methods. A number in parentheses indicates the number of methods surveyed for a particular method and automation type. The testing level for each method is represented as: minimal (blank), formal (F), informal (I) and model (M). Two methods support multiple levels of automation: Dome Tree visualization - log file analysis and simulation; and WebVIP - performance measurement and remote testing.

8. Philip J. Barnard. Cognitive resources and the learning of human-computer dialogs. In John M. Carroll, editor, *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, pages 112–158. The MIT Press, 1987.

9. F. Bodart, A. M. Hennebert, J. M. Leheureux, and J. Vanderdonckt. Towards a dynamic strategy for computer-aided visual placement. In T. Catarci, M. Costabile, M. Levialdi, and G. Santucci, editors, *Proc. of the International Conference on Advanced Visual Interfaces: AVI '94*, pages 78–87, Bari, Italy, 1994. ACM Press, New York.

10. Jose A. Borges, Israel Morales, and Nestor J. Rodriguez. Guidelines for designing usable world wide web pages. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 2, pages 277–278, 1996.

11. Neil Bowers. Weblint: quality assurance for the World Wide Web. In *Proceedings of the Fifth International World Wide Web Conference WWW5, May 6–10, Paris, France*. EPGL, 1996.

12. Michael D. Byrne, Bonnie E. John, Neil S. Wehrle, and David C. Crow. The tangled web we wove: A taxonomy of WWW use. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1, pages 544–551, 1999.

13. Michael D. Byrne, Scott D. Wood, Piyawadee "Noi" Sukaviriya, James D. Foley, and David Kieras. Automating interface evaluation. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1 of *Automatic Support in Design and Use*, pages 232–237, 1994.

14. Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

15. Ed H. Chi, Peter Pirolli, and James Pitkow. The scent of a site: A system for analyzing and predicting information scent, usage, and usability of a web site. In *Proceedings of ACM CHI 00 Conference on Conference on Human Factors in Computing Systems*, 2000.

16. David Clark and Daniel Dardailler. Accessibility on the web: Evaluation & repair tools to make it possible. In *Proceedings of Technology and Persons with Disabilities (CSUN'99)*, 1999. Available at http://www.dinf.org/csun_99/session0195.html.

17. Tim Comber. Building usable web pages: An HCI perspective. In Roger Debreceny and Allan Ellis, editors, *Proceedings of the First Australian World Wide Web Conference AusWeb'95*, pages 119–124. Norsearch, Ballina, 1995.

18. Michael Cooper. Universal design of a web site. In *Proceedings of Technology and Persons with Disabilities (CSUN'99)*, 1999. Available at http://www.dinf.org/csun_99/session0030.html.

19. J. Coutaz. Evaluation techniques: Exploring the intersection of HCI and software engineering. In *Proceedings of the International Conference on Software Engineering*, 1994.

20. John Cugini and Jean Scholtz. VISVIP: 3D visualization of paths through web sites. In *WebVis'99 International Workshop on Web-Based Information Visualization*, 1999.

21. Flavio de Souza and Nigel Bevan. The use of guidelines in menu interface design: Evaluation of a draft standard. In *Proceedings of IFIP INTERACT'90: Human-Computer Interaction*, pages 435–440, 1990.

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

23. Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-Computer Interaction*. Prentice Hall, 1993.

24. M. Carl Drott. Using web server logs to improve site design. In *ACM 16th International Conference on Systems Documentation*, pages 43–50, 1998.

25. Michael Etgen and Judy Cantor. What does getting WET (web event-logging tool) mean for web usability. In *Proceedings of The Future of Web Applications: Human Factors & the Web*, June 1999. Available at http://www.nist.gov/itl/div894/vvrg/hfweb/proceedings/etgen-cantor/index.html.

26. Pete Faraday and Alistair Sutcliffe. Providing advice for multimedia designers. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1, pages 124–131, 1998.

27. Peter Faraday. Visually critiquing web pages. In *Proceedings of EG Multimedia 99 Workshop*, 1999.

28. Christelle Farenc, Véronique Liberati, and Marie-France Barthet. Automatic ergonomic evaluation: What are the limits. In *Proceedings of the 2nd International Conference on Computer-Aided Design of User Interfaces (CADUI'96)*, Dordrecht, june 1999. Louvain-la-Neuve, Kluwer.

29. Christelle Farenc and Philippe Palanque. A generic framework based on ergomonics rules for computer aided design of user interface. In J. Vanderdonckt and A. Puerta, editors, *Proceedings of the 3rd International Conference on Computer-Aided Design of User Interfaces (CADUI'99)*, Dordrecht, October 1999. Louvain-la-Neuve, Kluwer.

30. Rodney Fuller and Johannes J. de Graaff. Measuring user motivation from server log files. In *Proceedings of the Human Factors and the Web 2 Conference*, October 1996. Available from http://www.microsoft.com/usability/webconf.htm.

31. F. A. Glenn, S. M. Schwartz, and L. V. Ross. 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, 1992.

32. Mark Guzdial, Paulos Santos, Albert Badre, Scott Hudson, and Mark Gray. Analyzing and visualizing log files: A computational science of usability. GVU Center TR GIT-GVU-94-8, Georgia Institute of Technology, 1994.

33. Monty L. Hammontree, Jeffrey J. Hendrickson, and Billy W. Hensley. Integrated data capture and analysis tools for research and testing on graphical user interfaces. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 431–432, 1992.

34. H. Rex Hartson, José C. Castillo, John Kelsa, and Wayne C. Neale. Remote evaluation: The network as an extension of the usability laboratory. In Michael J. Tauber, Victoria Bellotti, Robin Jeffries, Jock D. Mackinlay, and Jakob Nielsen, editors, *Proceedings of the Conference on Human Factors in Computing Systems : Common Ground*, pages 228–235, New York, April 13–18 1996. ACM Press.

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

36. Harry Hochheiser and Ben Shneiderman. Understanding patterns of user visits to web sites: Interactive starfield visualizations of WWW log data. Technical Report CS-TR-3989, University of Maryland, College Park, February 1999.

37. K. Holtzblatt and S. Jones. Contextual inquiry: A participatory technique for system design. In D. Schuler and A. Namioka, editors, *Participatory Design: Principles and Practice*, pages 180–193, Hillsdale, NJ, 1993. Lawrence Earlbaum.

38. James Hom. The usability methods toolbox. http://www.best.com/~jthom/usability/usable.htm, 1998.

39. Scott E. Hudson, Bonnie E. John, Keith Knudsen, and Mickael D. Byrne. A tool for creating predictive performance models from user interface demonstrations. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1999.

40. Human Factors Engineering. Usability evaluation methods. http://www.cs.umd.edu/~zzj/UsabilityHome.html, 1999.

41. International Standards Organization. Ergonomic requirements for office work with visual display terminals, part 11: Guidance on usability, 1999.

42. Melody Y. Ivory and Marti A. Hearst. Comparing peformance and usability evaluation: New methods for automated usability assessment. Unpublished manuscript, 1999. Available at http://www.cs.berkeley.edu/~ivory/research/web/papers/pe-ue.pdf.

43. R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling,*. Wiley-Interscience, New York, NY, USA, May 1991.

44. Robin Jeffries, James R. Miller, Cathleen Wharton, and Kathy M. Uyeda. User interface evaluation in the real world: A comparison of four techniques. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 119–124, 1991.

45. J. Jiang, E. Murphy, and L. Carter. Computer-human interaction models (CHIMES): Revision 3. Technical Report DSTL-94-008, National Aeronautics and Space Administration, May 1993.

46. Bonnie E. John and David E. Kieras. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transac-*

tions on Computer-Human Interaction, 3(4):320–351, 1996.

47. David J. Kasik and Harry G. George. Toward automatic generation of novice user test scripts. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Evaluation*, pages 244–251, 1996.

48. David Kieras and Peter G. Polson. An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22(4):365–394, 1985.

49. David E. Kieras, Scott D. Wood, Kasem Abotel, and Anthony Hornof. GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Evaluation, pages 91–100, 1995.

50. David E. Kieras, Scott D. Wood, and David E. Meyer. 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):230–275, September 1997.

51. Won Chul Kim and James D. Foley. Providing high-level control and expert assistance in the user interface presentation design. In Stacey Ashlund, Ken Mullet, Austin Henderson, Erik Hollnagel, and Ted White, editors, *Proceedings of the Conference on Human Factors in computing systems*, pages 430–437, New York, 1993. ACM Press.

52. Kevin Larson and Mary Czerwinski. Web page design: Implications of memory, structure and scent for information retrieval. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1, pages 25–32, 1998.

53. Andreas Lecerof and Fabio Paternó. Automatic support for usability evaluation. *IEEE Transactions on Software Engineering*, 24(10):863–888, October 1998.

54. Kenton Lee. Motif FAQ. http://www-bioeng.ucsd.edu/~fvetter/misc/Motif-FAQ.txt, 1997.

55. Rick Levine. Guide to web style. Sun Microsytems, 1996. Available at http://www.sun.com/styleguide/.

56. Clayton Lewis, Peter G. Polson, Cathleen Wharton, and John Rieman. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In *Proceedings of ACM CHI 90 Conference on Human Factors in Computing Systems*, pages 235–242, 1990.

57. Jonas Lowgren and Tommy Nordqvist. Knowledge-based evaluation as design support for graphical user interfaces. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 181–188, 1992.

58. Gene Lynch, Susan Palmiter, and Chris Tilt. The max model: A standard web site user model. In *Proceedings of The Future of Web Applications: Human Factors & the Web*, June 1999. Available at http://www.nist.gov/itl/div894/vvrg/hfweb/proceedings/lynch/index.html.

59. Patrick J. Lynch and Sarah Horton. *Web Style Guide: Basic Design Principles for Creating Web Sites*. Yale University Press, 1999. Available at http://info.med.yale.edu/caim/manual/.

60. Miles Macleod and Ralph Rengger. The development of DRUM: A software tool for video-assisted usability evaluation. In *Proceedings of the HCI'93 Conference on People and Computers VIII*, pages 293–309, 1993.

61. Rohit Mahajan and Ben Shneiderman. Visual & textual consistency checking tools for graphical user interfaces. Technical Report CS-TR-3639, University of Maryland, College Park, May 1996.

62. R. Molich, N. Bevan, S. Butler, I. Curson, E. Kindlund, J. Kirakowski, and D. Miller. Comparative evaluation of usability tests. In *Proceedings of UPA98*, pages 189–200, June 1998.

63. R. Molich, A. D. Thomsen, B. Karyukina, L. Schmidt, M. Ede, W. van Oel, and M. Arcuri. Comparative evaluation of usability tests. In *Proceedings of ACM CHI'99 Conference on Human Factors in Computing Systems*, pages 83–86, May 1999.

64. Jakob Nielsen. *Usability Engineering*. Academic Press, Boston, MA, 1993.

65. Dan. R. Olsen, Jr. *User Interface Management Systems: Models and Algorithms*. Morgan Kaufman Publishers Inc., San Mateo, CA, 1992.

66. Dan R. Olsen, Jr. and Bradley W. Halversen. Interface usage measurements in a user interface management system. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, pages 102–108, 1988.

67. Open Software Foundation. *OSF/Motif Style Guide*. Number Revision 1.1 (for OSF/Motif release 1.1). Prentice Hall, Englewood Cliffs, NJ, 1991.

68. Philippe Palanque, Christelle Farenc, and Rémi Bastide. Embedding ergonomic rules as generic requirements in the development process of interactive software. In A. Sasse and C. Johnson, editors, *Proceedings of INTERACT'99: IFIP TC13 Seventh International Conference on Human-Computer Interaction*, Edinburgh, Scotland, 1999. IOS Press.

69. Avraham Parush, Ronen Nadir, and Avraham Shtub. Evaluating the layout of graphical user interface screens: Validation of a numerical, computerized model. *International Journal of Human Computer Interaction*, 10(4):343–360, 1998.

70. F. Paternó, C. Mancini, and S. Meniconi. ConcurTaskTrees: Diagrammatic notation for specifying task models. In *Proceedings of INTERACT '97*, pages 362–369. Sydney: Chapman and Hall, 1997.

71. Fabio Paternó and Giulio Ballardin. Model-aided remote usability evaluation. In A. Sasse and C. Johnson, editors, *Proceedings of INTERACT'99: IFIP TC13 Seventh International Conference on Human-Computer Interaction*, pages 434–442, Edinburgh, Scotland, 1999. IOS Press.

72. C. A. Petri. Concepts of net theory. In *Mathematical Foundations of Computer Science: Proceedings of Symposium and Summer School*, pages 137–146, High Tatras, Czechoslovakia, 3–8 September 1973. Mathematical Institute of the Slovak Academy of Sciences.

73. R. W. Pew and A. S. Mavor, editors. *Modeling Human and Organizational Behavior: Application to Military Simulations*. National Academy Press, Washington, 1998. Available at http://books.nap.edu/html/model.

74. T. A. Polk and P. S. Rosenbloom. Task-independent constraints on a unified theory of cognition. In F. Boller and J. Grafman, editors, *Handbook of Neuropsychology, Volume 9*. Elsevier, Amsterdam, Netherlands, 1994.

75. Julie Ratner, Eric M. Grose, and Chris Forsythe. Characterization and assessment of HTML style guides. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 2, pages 115–116, 1996.

76. Matthias Rauterberg. From novice to expert decision behaviour: a qualitative modeling approach with petri nets. In Y. Anzai, K. Ogawa, and H. Mori, editors, *Symbiosis of Human and Artifact: Human and Social Aspects of Human-Computer Interaction*, volume 20B, pages 449–454. Elsevier Science, 1995.

77. Matthias Rauterberg. How to measure and to quantify usability of user interfaces. In A. Özok and G. Salvendy, editors, *Advances in Applied Ergonomics*, pages 429–432, West Lafayette, 1996. USA Publishing.

78. Matthias Rauterberg. A petri net based analyzing and modeling tool kit for logfiles in human-computer interaction. In *Proceedings of Cognitive Systems Engineering in Process Control CSEPC'96*, pages 268–275. Kyoto University: Graduate School of Energy Science, 1996.

79. Matthias Rauterberg and Roger Aeppili. Learning in man-machine systems: the measurement of behavioural and cognitive complexity. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics SMC'95*, pages 4685–4690. Institute of Electrical and Electronics Engineers, 1995.

80. H. Reiterer. A user interface design assistant approach. In Klaus Brunnstein and Eckart Raubold, editors, *Proceedings of the IFIP 13th World Computer Congress*, volume 2, pages 180–187, Amsterdam, The Netherlands, 1994. Elsevier Science Publishers.

81. John Rieman, Susan Davies, D. Charles Hair, Mary Esemplare, Peter Polson, and Clayton Lewis. An automated cognitive walkthrough. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, Demonstrations: Interface Design Issues, pages 427–428, 1991.

82. Dominique Scapin, Corinne Leulier, Jean Vanderdonckt, Céline Mariage, Christian Bastien, Christelle Farenc, Philippe Palanque, and Rémi Bastide. Towards automated testing of web usability guidelines. In *Proceedings of the 6th Conference on Human Factors & the Web*, 2000.

83. Jean Scholtz and Sharon Laskowski. Developing usability tools and techniques for designing and testing web sites. In *Proceedings of the 4th Conference on Human Factors & the Web*, 1998. Available at http://www.research.att.com/conf/hfweb/proceedings/scholtz/index.html.

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

85. Andrew Sears. AIDE: A step toward metric-based interface development tools. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 101–110, 1995.

86. Service Metrics. Service metrics solutions. http://www.servicemetrics.com/solutions/solutionsmain.asp, 1999.

87. Antonio C. Siochi and Deborah Hix. A study of computer-supported user interface evaluation using maximal repeating pattern analysis. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 301–305, 1991.

88. Sidney L. Smith. Standards versus guidelines for designing user interface software. *Behaviour and Information Technology*, 5(1):47–61, 1986.

89. Sidney L. Smith and Jane N. Mosier. Guidelines for designing user interface software. Technical Report ESD-TR-86-278, The MITRE Corporation, Bedford, MA 01730, 1986.

90. Lincoln D. Stein. The rating game. http://stein.cshl.org/ lstein/rater/, 1997.

91. Dennis J. Streveler and Anthony I. Wasserman. Quantitative measures of the spatial properties of screen designs. In *Proceedings of IFIP INTERACT'84: Human-Computer Interaction*, pages 81–89, 1984.

92. Terry Sullivan. Reading reader reaction: A proposal for inferential analysis of web server log files. In *Proceedings of the Human Factors and the Web 3 Conference*, June 1997. Available from http://www.uswest.com/web-conference/index.html.

93. Yin Leng Theng and Gil Marsden. Authoring tools: Towards continuous usability testing of web documents. In *Proceedings of the 1st International Workshop on Hypermedia Development*, 1998.

94. Harold Thimbleby. Gentler: A tool for systematic web authoring. *International Journal of Human-Computer Studies*, 47(1):139–168, 1997.

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

96. Dana L. Uehling and Karl Wolf. User action graphing effort (UsAGE). In *Human Factors in Computing Systems. Conference Proceedings CHI'95*. ACM, 1995.

97. Usable Net. LIFT online. http://www.usablenet.com, 2000.

98. J. Vanderdonckt and X. Gillo. Visual techniques for traditional and multimedia layouts. In T. Catarci, M. Costabile, M. Levialdi, and G. Santucci, editors, *Proc. of the International Conference on Advanced Visual Interfaces: AVI '94*, pages 95–104, Bari, Italy, 1994. ACM Press, New York.

99. Web Accessibility Initiative. Web content accessibility guidelines 1.0. World Wide Web Consortium, Geneva, 1999. Available at http://www.w3.org/TR/WAI-WEBCONTENT/.

100. Web Criteria. Max, and the objective measurement of web sites. http://www.webcriteria.com, 1999.

101. Andy Whitefield, Frank Wilson, and John Dowell. A framework for human factors evaluation. *Behaviour and Information Technology*, 10(1):65–79, 1991.

102. World Wide Web Consortium. HTML validation service, 2000. Available at http://validator.w3.org/.

103. Yahoo. Html validation checkers, 2000. Available at http://dir.yahoo.com/Computers_and_Internet/Information_and_Documentation/Data_Formats/HTML/Validation_and_Checkers/.

104. Richard M. Young, T. R. G. Green, and Tony Simon. Programmable user models for predictive evaluation of interface designs. In *Proceedings of ACM CHI'89 Conference on Human Factors in Computing Systems*, pages 15–19, 1989.

105. W. Zachary, J.-C. Le Mentec, and J. Ryder. Interface agents in complex systems. In C. N. Ntuen and E. H. Park, editors, *Human Interaction With Complex Systems: Conceptual Principles and Design Practice*. Kluwer Academic Publishers, 1996.

106. Panayiotis Zaphiris and Lianaeli Mtei. Depth vs. breadth in the arrangement of Web links. http://www.otal.umd.edu/SHORE/bs04, 1997.

107. Luke S. Zettlemoyer, Robert St. Amant, and Martin S. Dulberg. IBOTS: Agent control through the user interface. In *Proceedings of the 1999 International Conference on Intelligent User Interfaces*, pages 31–37, 1999.