A proposed architecture for integrating accessibility test tools

P. Englefield
C. Paddison
M. Tibbits
I. Damani

Automated test tools are an essential resource for practitioners responsible for evaluating the accessibility of Web sites. However, both systematic analysis of tool capabilities and practitioner feedback have identified a range of practical issues that mar the effectiveness of existing tools. In practice, although automated test tools need to be used in combination to give good coverage, their lack of consistent user experience and their diverse reporting formats discourage such combined usage. Furthermore, test tools are expensive to develop; in addition to core analytical capability, authors must individually construct the user interface, I/O routines, Web crawlers, and report writers. In this paper, an architecture is proposed to address these concerns. In this architecture, tools are developed as plug-ins to an infrastructure that provides a common user interface, crawling and parsing services, and practitioner-oriented tools for analysis and reporting. The architecture supports an efficient, systematic evaluation process and benefits accessibility practice in two distinct ways: first, it simplifies the task of the evaluator by providing a consistent, integrated, and efficient user experience for executing, reporting, and communicating a study; second, it supports an economic model in which tools can release development resources from mundane software engineering activities in order to invest in the intelligent-agent development necessary to address the deeper challenges of automated testing.

INTRODUCTION

Accessible design is intended to enable universal access[1] to interactive systems, regardless of user impairments and preferred client technology. Such design supports the specific needs of distinct groups challenged by impairments related to vision, hearing, motor skills, and cognitive abilities. To use Shneiderman's definition, "Universal usability will be met when affordable, useful, and usable technology accommodates the vast majority of the global population: this entails addressing challenges of technology variety, user diversity, and gaps in user knowledge in ways only beginning to be acknowledged by educational, corporate, and government agencies."[2]

Although the methods of accessible design are well understood, a majority of existing Web sites fail to meet the fundamental needs of people with disabilities. For example, a 2004 study carried out by City University on behalf of the UK Disability Rights Commission[3] found that less than 19 percent of all home pages and only 32 percent of government home pages conformed to level A, the most basic level of compliance defined by W3C** (World Wide Web Consortium) Web Content Accessibility Guidelines (WCAG).[4] (The WCAG guidelines define three levels of compliance, with "A" being the lowest and "AAA" the highest).

■ A majority of existing Web sites fail to meet the fundamental needs of people with disabilities ■

This paper is organized as follows. In the next section we provide a background on accessibility design issues, existing automated-test-tool technology, and our proposed architecture for automated test tools. In the following section, the details of current test tool characteristics and adherence to guidelines are described. We then summarize feedback from practitioners on current tools. The next section outlines design criteria for our test tool architecture and is followed by a presentation of the architecture in detail. The paper closes with a summary of our work and an indication of future research directions.

BACKGROUND

Design for accessibility encompasses a complex set of requirements. In particular, because of the diverse needs involved, testing for compliance with accessibility requirements can be difficult and time consuming. Automated test tools could make the situation easier, but current tools exhibit a series of issues that make them less than ideal. In the following sections we discuss these questions in more detail and then describe our proposed architecture intended to address these concerns.

Design for accessibility

A diverse range of challenges must be addressed in design for accessibility. For example, distinct design responses are necessary to support blind users, as compared to the support required for users with other visual impairments such as limited vision or tunnel vision. In the former case, a key concern is providing appropriate encoding of content[5] for screen readers, such as IBM Home Page reader.[6] (Technologies for Disabilities Information Service [TechDis][7] defines a screen reader as "a software program that reads the contents of the screen aloud to a user . . . usually used by blind and visually impaired people. Screen readers cannot read text that is part of an image.") In the case of visual impairments, the emphasis is on a range of techniques, such as appropriate typography, sensitivity to the diminished context associated with use of a screen magnifier, and support for user-defined font sizes.[8] Additional design responses are required to facilitate input by users with limited dexterity, strength, or mobility, users with hearing impairments, and users with specific or general learning difficulties. Moreover, disabilities are diverse in degree. Universally usable design recognizes not only the needs of screen reader users but also the very real requirements of those whose eyesight and dexterity are somewhat diminished by age or by acquired conditions such as carpal tunnel syndrome.

Although awareness of disability is critical to universal usability, designers should not see disability as the primary defining characteristic of an audience. Users of assistive technology are primarily citizens, employees, and social actors, and only secondarily people living with a disability. Section 508 of the U.S. Rehabilitation Act defines assistive technology as a "device or software that substitutes for or enhances the function of some impaired ability."[9] In this sense, design for accessibility is a special case of design for usability. The IBM User Engineering process,[10] for example, recommends an initial focus in design on goals and constraints. The goals for accessible design are likely to be similar to those for nondisabled users, namely research, communication, learning, buying, and other transactions related to physical, social, and existential needs.[11] The difference comes in the constraints. A critical factor in design for accessibility is design for assistive technology such as screen readers, adapted keyboards, specialized pointing devices, and built-in support within browsers and operating-system user interfaces. From an abstract design process perspective, this can be seen as analogous to design for heterogeneous clients such as WAP (Wireless

Application Protocol) phones and PDA (Personal Digital Assistant) devices; each device offers capabilities and constraints that the designer must understand and respond to.

Because assistive technologies rely on guideline-compliant content, accessibility advocates recommend that content developers respect these guidelines during implementation. For example, formatting text in HTML (Hypertext Markup Language) rather than using style sheets confounds the user-defined preferences in a browser and makes the user experience more difficult than it needs to be. Similarly, a text label rendered as a JPEG (Joint Photographic Experts Group) image is opaque to a screen reader without additional tagging.

Although compliant implementation is critical to success, more challenging issues must be resolved earlier in the design process. In initial design, for example, so-called vast-and-fast menus[12] support recognition over recall for sighted users,[13] but for users of screen readers this design pattern imposes an unacceptable burden on working memory. Likewise, during detailed design, visual designers may need to make trade-offs between contemporary typographic aesthetics that value compact, low-contrast typography and the needs of readers with low vision.

Rittel[14] distinguishes *tame* and *wicked* problems. Tame problems can be analyzed by established methods and have a single and recognizable solution. In contrast, wicked problems are hard to define to the satisfaction of all stakeholders, have no clear stopping rules (rules to determine when sufficient work has been done), have better or worse (rather than right or wrong) solutions, have no objective measure of success, have no given alternative solutions, and often have moral, political, and professional dimensions. Tame problems are solved by rigorously following a specified procedure, but wicked problems are addressed by argumentation. Whereas checking for compliance with guidelines is a tame and tractable problem, problems of the class described here for accessibility design are wicked problems and can only be solved by argumentation and designer insight.

Design for accessibility offers value to both users and stakeholders. In particular, although acknowledging the poor quality of much design,[3] some blind users nevertheless see the Web as fundamentally enabling.[15] For stakeholders, accessible design

■ Design for accessibility is a special case of design for usability ■

supports organizational goals related to both business development and risk avoidance. For example, in the UK, people with disabilities represent both a significant marketplace and also a large pool of potential employee talent. In fact, according to a disability briefing published by the UK Disability Rights Commission,[16] nearly one in five people of working age is disabled, with disabled people only half as likely as nondisabled people to be in employment.

Design for accessibility is also a regulatory compliance issue because many countries have implemented aggressive legislation to ensure reasonable access to services. Examples include the UK Disability Discrimination Act[17] and Section 508 of the U.S. Rehabilitation Act.[18]

Testing for accessibility

A distinct feature of contemporary user-experience design methods is an emphasis on rigorous evaluation of design proposals.[10,19] Empirical testing with users[20] is frequently preferred because of its sensitivity and evidential weight. However, inspection methods, such as Nielsen's heuristic evaluation,[13] are widely used in commercial work as a result of their lower cost, faster turnaround,[21] and technical focus.

Both of these methods are used in evaluating the user experience for users with disabilities. Given the importance of standards compliance, however, testing for accessibility also typically involves automated test tools. Such tools *crawl* through a Web site and identify various coding solecisms, such as malformed HTML or the absence of tags essential to assistive technology. Automated test tools are relatively naive. For example, they can reliably detect the presence or absence of an HTML <ALT> attribute but are unable to form a judgment as to whether the tag's content is helpful. Test tools are not restricted to HTML validation; for example, IBM

Easy English Analyzer[22] is helpful for assessing the reading difficulty of English text.

Advocacy groups tend to prefer empirical evaluation. For example, a recent report from the UK

■ Design for accessibility offers value to both users and stakeholders ■

Disabilities Rights Commission strongly supported the need to involve disabled people in scientific testing of Web designs.[3] The RNIB (Royal National Institute of the Blind) likewise strongly endorses this approach.[23] However, in commercial practice, evaluation typically involves a considered combination of user testing, expert inspection, and automated testing, the relative mix of which depends on deadlines, funding, and availability of participants with the required class and degree of disability. Thus, in a recent study for a UK local government project,[24] evaluators were required to deliver a rapid assessment of the accessibility of a wide range of design proposals and chose to combine automated testing and inspection to maximize the scope of the evaluation.

Issues with automated test tools

Although automated test tools are an essential component of the evaluation toolkit, a range of issues with these tools can be identified that impact both adoption by practitioners and also development by subject matter experts. In order to carry out a full-spectrum evaluation, tools must be used in combination. However, these tools generally do not interact gracefully with one another. Inconsistent inputs, incompatible outputs, ambiguous warrants, and patchy mutual coverage add to the burden of using these resources in combination. (The term *warrant* is used in this paper to mean an endorsement from a reputable organization typically recognized as an authority on a given topic.) Specific tools issues include:

1. The cost of learning diverse user interfaces
2. The difficulty of physically combining input from different tools
3. The challenge of aggregating output data based on incompatible conceptual models
4. The lack of standards for defining test capabilities and the authorities that provide a warrant for those capabilities
5. The need to run tools individually and sequentially

In addition to the core capability of assessing content against some set of guidelines, each tool developer must also design and implement a user interface, I/O routines, and services to crawl and parse Web content. Much of this expensive infrastructure is common to the majority of tools, but experienced IBM developers estimate that creating this infrastructure consumes 40–50 percent of the development effort for a given application. Moreover, many tools authors belong to noncommercial, charitable, or academic organizations. For these groups the cost of developing such peripheral technology is likely either to discourage the undertaking of tools projects or to divert skilled resources from development of core functionality related to compliance with accessibility guidelines. Commercial developers may also be affected by the need to make substantial investment in areas beyond their core competence, potentially leading to higher pricing.

A proposed architecture for automated test tools

The preceding analysis suggests four essential issues that impact both tool users and tool makers: *consistency, integration, authority*, and *development cost*. We propose here an architecture to explicitly address these concerns. Although this proposal specifically addresses concerns related to accessibility tooling, the approach might also usefully be generalized to support automated testing for usability, application development, and other inspection-based disciplines.

The architecture has four components:

1. A set of *standards* provides coherent classes and definitions for tool capabilities, guidelines against which compliance assessments are made, supporting warrants, and logical formats for outputs.
2. A common *user interface* offers a single, integrated, consistent user experience to enable practitioners to design, execute, and report any test that exploits a set of tools.
3. An enabling *framework* provides shared services for input and output, and a mechanism to run and monitor tools in parallel.

4. A defined *client interface* enables tools to be registered and integrated in the framework.

Each tool need only implement core function to advertise and execute its capabilities to assess HTML with respect to a set of guidelines. In particular, this architecture mandates no specific enabling technology. For example, it might equally well be implemented by using Web services[25] or by using Eclipse.[26] The need for consistency is explicitly supported by a common user interface and a framework service for writing output in a standard physical and logical format. Although consistency is delivered by technology, it is underwritten by standards.

The goal of integration is satisfied by the framework. A diverse set of tools is installed, selected, customized, and invoked by using a single user interface. The output from multiple tools is aggregated and presented as a whole for review, analysis, and reporting. The requirement for authority is handled by requiring each tool to support its advertised capabilities with respect to both authoritative guidelines and warrants. Finally, the issue of development cost is addressed by a framework that provides a rich set of services, so that individual tool developers have no need to build code for a user interface, file management, Web crawling, and content parsing.

Many cases of commercially successful plug-in models for specialized tools can be identified. Examples include browser plug-ins such as Acrobat** viewer and Flash**, browser extensions such as the Google toolbar**, Photoshop** filter plug-ins for visual treatment of artwork, and third-party extensions to games such as Microsoft Flight Simulator.

Clearly, a well-designed framework does not in itself ensure widespread adoption. Examples of good designs that never achieved widespread success include the IBM San Francisco project,[27] the Betamax** video format, and APS (Advanced Photo System) format photographic stock. Potential inhibitors to success include performance, scalability, timing, and promotion. The existence of a strong architecture with compelling benefits is likely to be a necessary but not sufficient condition for adoption.

Rogers[28] proposes a standard set of criteria for diffusion of innovation in which an adoptable innovation delivers the following:

1. *Relative advantage*—From the user's perspective: "What's in it for me?"
2. *Compatibility*—"Does it fit my values and practices?"
3. *Simplicity*—"Can I easily understand the value statement?"
4. *Suitability for tryout*—"Can I try it before I commit to use it?"
5. *Observability*—"Is it easy to see the claimed benefits?"

The proposed framework meets criteria 1, 3, and 5 from this list. Criterion 2 needs to be dealt with by the design of technical detail and standards. Criterion 4 might be addressed by appropriate commercial packaging.

THE TEST TOOLS LANDSCAPE

This section provides an overview of current automated test tools. In particular, we focus on tool characteristics and compliance with established guidelines.

Methodology

The W3C WAI (Web Accessibility Initiative)[29] provides a useful general classification of tools as follows:

1. *Evaluation tools*—These tools perform a static analysis of pages or sites regarding their accessibility and return a report or a rating.
2. *Repair tools*—After the accessibility issues with a Web page or site have been identified, these tools can assist the author in making the pages or site more accessible.
3. *Filter and transformation tools*—These tools assist Web users rather than authors to either modify a page or supplement an assistive technology or browser.

A more specific classification of *evaluation* tools can be done as follows:

a) *General*—Tools that perform tests for a variety of accessibility issues
b) *Focused*—Tools that test for one aspect or a limited aspect of accessibility

c) *Services*—Tools that run on an ongoing basis, such as proxies, Web services, and monitors.

Based on this classification scheme, a set of *general* and *focused* evaluation tools was selected for review from a mix of sources including commercial software developers, academic institutions, advocacy organizations, and internal IBM teams. Initially the descriptions of 30 tools were taken from Web sites and analyzed to explore general coverage and consistency. A representative set of seven tools was then installed and analyzed for detailed compliance with specific WAI guidelines.

■ The existence of a strong architecture with compelling benefits is likely to be a necessary but not sufficient condition for adoption ■

Inputs, parameters, and outputs

The majority of tools reviewed were designed to work with Web sites. The use of input formats in these tools is reasonably homogenous and can be considered in terms of a general case of Web site analysis extended by necessary specializations to analyze specific file types. A subset of tools was designed to work with specialized file types such as image files, text elements, and specific markup such as XML (Extensible Markup Language), XHTML (Extensible Hypertext Markup Language), SVG (Scalable Vector Graphics), and JSP** (JavaServer Pages**) files.

Generic parameters include a starting point and a set of options. Although the majority of tools specify the starting point as a URL (Uniform Resource Locator), a few require a domain name. More specialized parameters include assessed guidelines, assessed impairment, and choice of language.

Output formats show a high level of diversity in both logical and physical format. Some tools provide summaries, and others offer detailed reports. Some tools provide reports as printed output, others generate reports in HTML or Lynx[30] format, and some update the pages in which errors are found. In addition to the inconsistency in medium, reports are not written according to any common conceptual model, and data from different tools cannot easily be compared and aggregated.

General guideline coverage

Table 1 illustrates the stated coverage of various sets of guidelines for all 30 tools. For example, although 50 percent of the tools support the WCAG guidelines, only 23 percent support all three WCAG priority levels. A further 50 percent suggest that they support WCAG but do not explicitly state which guidelines are supported.

WAI guideline coverage

Table 2 shows the degree to which the WAI guidelines are covered by the seven tools selected for detailed analysis. Thus, WAI guidelines 1.1 and 5.1 are assessed by all seven tools, but guideline 3.1 is not assessed by any of the seven.

WAI organizes accessibility guidance as sets of checkpoints within guidelines. Each of the 14 guidelines includes between one and ten checkpoints. *Table 3* shows the degree to which the seven tools claim to test each of the individual checkpoints. The columns labeled 1 through 10 show the proportion of the seven tools that assess each checkpoint. For example, although all seven tools inspect checkpoint 1 of guideline 1, no tools assess checkpoint 7 of guideline 3. A dash indicates that the guideline has no checkpoint for a given number. (For example, guideline 1 has only five checkpoints.) The final column gives the mean coverage for all checkpoints within a guideline.

Analysis

The consistency of input formats reflects the common goal of these tools, namely to analyze Web pages. However, there is some diversity in the inputs required by more specialized tools. By contrast, logical and physical output formats are highly mutually inconsistent.

Some tools, but not a majority, define their capabilities in terms of guidelines such as WCAG, Section 508, and HTML. However, there is a high variation in the precise subsets of these guidelines that each such tool chooses to support. In many cases tools do not explicitly state the guidelines for which they test. Coverage of guidelines and checkpoints is extremely variable; some are assessed by many tools, others by few. In fact Knight reports a

Table 1 Coverage by guideline

Guidelines		Number of Tools Stating Support for the Guideline	Percentage of Tools Stating Support for the Guideline
WCAG	WCAG 1.0 (Priority 1, 2, and 3)	7	23
	WCAG 1.0 (Priority 1 and 2)	0	0
	WCAG 1.0 (Priority 1)	1	3
	WCAG 1.0 (priority level unspecified)	6	20
	WCAG 2.0 (priority level unspecified)	1	3
	Unstated or unobtainable	15	50
Section 508	All Section 508	12	40
	Unstated or unobtainable	18	60
HTML	HTML 4.0	2	7
	HTML 3.2 Standard	1	3
	HTML 1.1	1	3
	HTML (version unspecified)	3	10
	Unstated or unobtainable	23	77
IBM Guidelines	Web	2	7
	Unstated or unobtainable	28	93
Other	Brinck	1	3
	CLF (Common Look and Feel)	1	3
	Diamond-Bullet Syntax	1	3
	Gergle Wood	1	3
	JIS (Japanese Industrial Standards)	1	3
	SMOG (Simple Measure of Gobbledegook)	1	3
	XAG (XML Accessibility Guidelines)	1	3
	CSS (Cascading Style Sheet)	1	3
	Unstated or unobtainable	22	73

comment from Wise[31] that "Tools are lacking or horribly expensive for AAA compliance."

ANALYSIS OF PRACTITIONER EXPERIENCE

In addition to doing actual tool analysis, we also sought out commentary from practitioners who use these tools. This section describes the methodology, results, and analysis of this part of our study.

Methodology

A small-scale informal study was carried out to acquire formative knowledge of current accessibility testing practice as well as perceived issues and requirements. A group of four IBM accessibility practitioners completed a questionnaire designed to probe their experiences using automated test tools. Additionally, three specialists from industry, academia, and an advocacy organization were interviewed to explore specific issues raised by the initial survey.

Practitioner feedback

Participants reported using tools in combinations to, in their words, "provide different perspectives" and "check different types of applications." For example, one participant reported using Inspect32 and AccExplorer32[32] in tandem. Another reported using LynxView[33] for a quick overview and LIFT Online[34] for an in-depth audit. In a third case, LIFT[35] was seen as useful for identifying technical problems,

Table 2 Degree of coverage of WAI Guidelines

Guidelines Covered	Number of Tools Stating That They Test These Guidelines	Percentage of Tools Stating That They Test These Guidelines
1.1, 5.1	7	100
6.3	6	86
1.2, 2.1, 3.3, 5.2, 6.1, 7.2, 7.3	5	71
1.4, 3.4, 4.3, 5.5, 6.5, 12.4	4	57
1.3, 1.5, 2.2, 3.2, 3.6, 7.1, 7.4, 7.5, 9.3, 10.2, 11.2, 12.2, 13.1, 13.2, 13.6, 13.9	3	43
3.5, 3.7, 4.1, 4.2, 5.3, 5.6, 6.4, 8.1, 9.1, 9.4, 9.5, 10.1, 10.5, 11.4, 12.1, 12.3, 13.3, 13.7, 14.3	2	29
5.4, 6.2, 9.2, 10.3, 10.4, 11.1, 11.3, 13.4, 13.5, 13.8, 13.10, 14.1, 14.2	1	14
3.1	0	0

but another tool was used to discover issues related to the use of color.

In this feedback, lack of consistency and lack of integration were highlighted as particular problems. For example, one participant stated, "There is a lot of replication across different tools. It would be very good if they were packaged together." Another participant stated a requirement for one tool that is automated and can test for all disability software parameters, stating, "Right now, in order to test for the entire range of compliance, multiple tools need to be used."

The participants also expressed a number of concerns regarding the usability and quality of current tools and the effectiveness of generated reports. Specifically, they found reports to be "overwhelming," difficult to use, too large, unhelpful as a medium for communicating findings to managers

Table 3 Degree of coverage of checkpoints within WAI guidelines

Guideline	Percentage of Coverage of Checkpoints										Overall Percentage of Coverage
	1	2	3	4	5	6	7	8	9	10	
1	100	71	57	43	43	-	-	-	-	-	63
2	71	43	-	-	-	-	-	-	-	-	57
7	71	71	43	43	43	-	-	-	-	-	54
6	86	71	57	29	14	-	-	-	-	-	51
12	57	43	29	-	-	-	-	-	-	-	43
5	71	57	29	14	29	-	-	-	-	-	40
4	57	29	29	-	-	-	-	-	-	-	38
3	71	57	43	29	29	29	0	-	-	-	37
8	29	-	-	-	-	-	-	-	-	-	29
9	43	29	14	29	29	-	-	-	-	-	29
13	14	14	14	32	43	43	43	29	29	14	29
10	43	29	14	14	29	-	-	-	-	-	26
11	43	29	14	14	-	-	-	-	-	-	25
14	14	14	29	-	-	-	-	-	-	-	19

and developers, and missing practical advice on fixing problems. One participant observed, "The way I work involves moving between different applications and sites and [the tools] all present results in different ways, so it gets complicated." Another commented that a standardized reporting system would be ideal to reconcile reports of the same issue by different tools. Other concerns included quality, overly technical interfaces, and insufficient support for customization.

The currently available tool set is also perceived to be mutually incomplete. For example, participants were concerned that there was little support for testing font sizes and high-contrast-mode issues. One specifically expressed a need for a tool to assess tab order. On the other hand, participants valued certain specific abilities of current tools, including facilities to inspect the code in error, parameters to adjust the level of compliance to be tested, and suggested fixes to erroneous source code.

Participants described the following requirements for test tools:

1. They should be integrated in the development environment to minimize learning and simplify use.
2. They should produce consistent reports to support intraproject comparisons.
3. They should report findings in HTML and provide "nice and flexible" facilities to "reorder in nice different ways."
4. They should provide graphic summaries, such as pie charts.
5. They should be flexible and capable of accommodating new capabilities.

One participant looked for a system that would do the following: "Bring together all the tools"; "Take you through the evaluation process"; "Run the automated tests for you"; and finally, "When you press the button at the end, it gives you a report that you can send to your client [that] would identify issues and tell [the client] what they need to do."

Analysis

This research suggests that although practitioners do use tools in combination, they are hindered by a lack of consistency, integration, and mutual coverage. Reports are seen to lack specificity, consistency, and relevance for target audiences. An integrated, process-oriented framework, closely tied to development environments, can be identified as a desirable strategy for addressing these concerns. *Table 4* identifies the support such a framework would provide for each of these questions.

Overall, issues concerning testing tools can be separated into two main categories:

1. Ergonomic issues related to the practical deployment of test tools by evaluators
2. The analytical power of the algorithms used by these tools

Table 4 illustrates the value of the proposed architecture as an explicit response to the first set of issues. Although this work does not claim to specifically address the second set of issues, an implementation might well support an economic model that encourages more investment in intelligent-agent-based tools. For example, commercial developers might then be able to reassign development resources previously consumed in developing user interface components to the implementation of more advanced analysis algorithms. Additionally, a common platform would offer a robust and attractive implementation channel for innovative algorithms developed by academic researchers.

DESIGN PRINCIPLES

Analysis of the practitioner feedback and current tool characteristics suggests the following guidelines for an effective architecture. In particular the architecture should:

1. *Demarcate responsibilities*—Common services should be handled by the architecture and unique and specialized services by the tools.
2. *Deliver consistency through tools not rules*—The architecture must deliver an integrated user experience and consistent reports through a standards-based interface to tools.
3. *Adapt to research goals*—Practitioners must be able to specify the scope of the test in terms of Web pages, classes of disability, and levels of compliance.

Table 4 Framework support for issues identified by practitioners

Reported Issue	Level of Support	Nature of Support
A. Lack of consistency	High	The framework provides a common user interface, a shared taxonomy of capabilities, and a standard reporting format.
B. Lack of integration	High	The set of pluggable tools are presented to the evaluator as a single integrated package.
C. Reports hard to use	High	Filtering, sorting, and cross-tabulation support structured analysis of results.
		The framework supports a range of output formats to meet specific audiences and purposes.
		Exporting results as XML enables the use of XSL to design tailored reports for specialized purposes.
D. Reports too large	High	Filtering enables reports to eliminate 'noise' to emphasize data relevant to a specific study.
E. Reports unhelpful for communicating results to stakeholders	High	See C.
F. Inconsistent reporting formats	High	The framework provides and enforces a common logical and physical output format.
G. Hard to reconcile reports from different tools	High	Output from all tools is integrated into a common format. Sorting enables convenient comparison. Where tool capabilities overlap, a single capability in the less-preferred tool can be selectively disabled.
H. Quality	Medium	Quality issues arising from development costs are mitigated by eliminating costs associated with development of user interface and I/O. Quality issues related to the user interface and I/O routines are eliminated. Issues associated with deep problems in artificial intelligence are not addressed other than by the economic argument for K.
I. Overly technical interfaces	Medium	A common user interface can be developed to a high standard of usability by applying IBM User Engineering techniques. The costs of this activity are incurred once and the benefits realized for each tool.
J. Insufficient support for customization	Low	The framework encourages developers to support a standard set of customization parameters.
K. Tool set mutually incomplete	Medium	The framework provides an economic model to encourage the development of specialized tools. It enables both commercial and academic organizations with skills in agent design to exploit their core capabilities without the need to invest in mundane software engineering activities.
L. The tools run and report all tests together	High	The framework provides a transport mechanism to run a selected subset of tools as a single operation and consolidate the generated output.

4. *Support practitioners*—Minimize effort and maximize effectiveness and satisfaction of practitioners (c.f., International Organization for Standardization (ISO) Standard 9241[36]).
5. *Support evaluation stakeholders*—Ensure that reports are useful to managers, designers, and developers.
6. *Support tools authors*—Minimize the development costs for tools; maximize the likelihood of adoption.

THE ARCHITECTURE

Figure 1 summarizes the proposed architecture for automated test tools. The next sections explain this architecture in more detail.

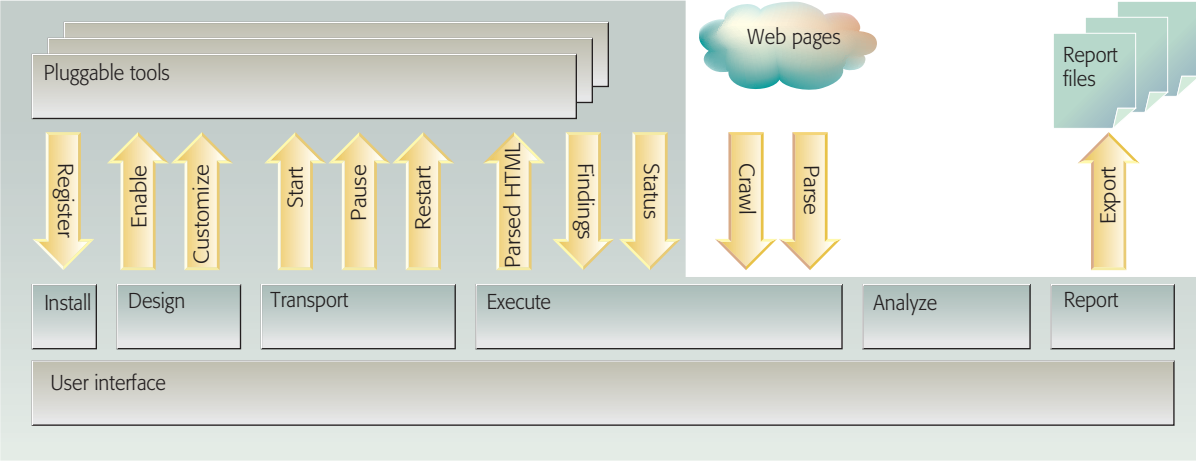

Figure 1
Overview of proposed architecture for automated test tools

Overview of the architecture

In this architecture an infrastructure provides facilities to *install* additional tools, *design* a study, pause and restart tests (*transport*), handle scheduling and I/O (*execute*), *analyze* findings, and export *reports* in a standard format. ***Table 5*** and ***Table 6*** give an overview of the proposed architecture in terms of the *standards, user interface components, infrastructure services*, and *client tool services* that apply to key life-cycle *events*. Table 5 describes the events involved in initially installing tools. Table 6 lists the events that would take place within a typical study.

In this architecture, standards provide coherent classes and definitions for tool capabilities, guidelines against which compliance assessments are made, supporting warrants, and logical formats for outputs. During installation, pluggable tools call infrastructure services to *register* their capabilities. To design a study, a practitioner uses a common user interface both to selectively enable and disable individual tool capabilities and to specify the values of runtime parameters required by tools. Client tool services are then implemented by the tool builder. To run a study, the practitioner uses the common transport interface to start, pause, and restart the test, and also to check the status of running tools. A checkpoint/restart capability enables large complex tests to be suspended and restarted at a convenient time. During execution, the framework uses shared services to crawl and parse Web pages and pass tokenized content to each enabled tool. Each tool analyzes the parsed content by using its own algorithms and rules and returns findings to the infrastructure in a canonical format. Tools also respond to scheduled requests for status from the infrastructure in order to display progress in the common user interface. A common

Table 5 Architectural overview for setup

	Standards	Common User Interface	Infrastructure Services	Client Tool Services
Install	Platform standards for creating plug-ins	• Install • Uninstall • Refresh	• Install • Uninstall • Refresh	As required by platform standards for plug-in
Register (enables the tool author to identify the capabilities of the tool)	• Taxonomy of capabilities: Selectable evaluation criteria (e.g., A, AA, AAA) • Taxonomy of standard parameters • Taxonomy of data types for custom parameters	n/a	Request and aggregate registration data	On request, register name, author, capabilities, warrants, standard parameters, and custom parameters

Table 6 Architectural overview for usage

	Standards	Common User Interface	Infrastructure Services	Client Tool Services
Design (enables the practitioner to select which tools to use and to supply required runtime parameters)	• Taxonomy of capabilities • Taxonomy of standard parameters • Taxonomy of data types for custom parameters	• List aggregated capabilities • Sort and filter by capability, warrant, name, and author. • Selectively enable and disable individual capabilities • Select preferred tool in the case of overlaps • Specify values for standard parameters. • Specify values for custom parameters. • Save and reuse designs	• Selectively enable and disable capabilities within tools • Pass standard and custom parameter values to tools	n/a
Transport (enables the user to start, monitor, pause, and cancel the test)	n/a	• Start • Pause • Restart • Abandon	n/a	• Start • Pause • Write checkpoint • Restart from checkpoint • Abandon
Execute (runs the selected tools in a controlled and efficient manner)	Logical output format	Report consolidated status	• Crawl • Parse • Notify tools about new page for analysis • Write aggregated findings in standardized format	• Report status on request • Request unit of content • Notify about findings
Analyze (enables the practitioner to review and analyze the aggregated findings)	n/a	• List, sort, filter • Search • Edit • List actions	n/a	n/a
Report (exports selected findings to external files for communication with stakeholders)	Physical file formats	Select or specify file and format	n/a	Export to selected file and format

user interface enables the practitioner to review and edit the aggregated findings and export one or more reports in a standardized format.

Standards and services

The following sections discuss in more detail the specific standards and services of this architecture for each of the event types.

The install event

This event describes the standards and services required to enable a practitioner to install an additional tool in the infrastructure. The infrastructure must be able to install, uninstall, and refresh tools. In particular, the process of refreshing enables tools to incorporate updated assets such as analytical rules.

Any implementation of the proposed test tools architecture is necessarily based on an existing platform architecture, such as Web Services or Eclipse. This implementation is likely to provide standards-based services for installing pluggable components. For example, when Eclipse is used as a platform, each tool must be packaged as a set of Java** classes and XML elements that are written to a standardized form prescribed by the platform. Eclipse itself provides a user interface for installing and uninstalling plug-ins and for refreshing resources from a server.

The register event

This event describes the standards and services required to enable tools developers to register the capabilities that each tool can contribute to the framework. Automated test tools typically provide capabilities to test only specific aspects of accessibility; in practice, the architecture described here encourages the development of specialized tools with a narrow focus. To work well in combination, tools need to register their attributes in a canonical form that can be conveniently aggregated with the attributes of other tools. Specifically, each tool must register the following attributes:

1. *Identity*—This includes the tool's name and version together with the name and logotype of the organization responsible for its development.
2. *Capabilities*—These define both the accessibility factors that the tool can assess and the warrants that back its claims for effectiveness. They are specified in terms of a controlled vocabulary defined by a formal taxonomy. This ensures that individual tools describe themselves in a standard way and are grouped consistently and logically within the user interface. Ideally, tools with a broad scope would be constructed on a granular basis so that individual capabilities could be registered and used separately.
3. *Parameters*—Both standard and custom parameters should be specified.

An appropriate taxonomy of test capabilities might usefully reference both outcome and scope. An outcome describes the tool's ability to identify errors with respect to a formal standard of compliance. For example, a tool might claim to detect class A and AA errors as defined by the W3C guidelines. In contrast, scope describes the facets of accessibility that the tool claims to test. For example, individual tools might focus on support for blind users, arthritic users, or dyslexics. Ideally, such a classification should be hierarchical to enable practitioners to aggregate tools and capabilities at various levels of abstraction. Thus, a hierarchical scheme might decompose high-level classes such as perceptual disabilities, motor disabilities, and cognitive disabilities. Existing proposals for taxonomies of disability may prove to be useful resources. Examples include the functional limitation model[37] and the World Health Organization model.[38] Warrants are expressed as a list of endorsing organizations, such as advocacy groups and government bodies.

Many existing tools permit the user to enter parameters to specify aspects of the test. For example, tools that crawl a Web site typically invite the user to enter the URL of the starting point and indicate the required depth of pages to be processed. The architecture proposes two standards related to parameters: first, a nomenclature of standard parameters that many tools will use, and second, a set of data types for custom parameters unique to a specific tool. Examples of the former might include *start page, crawling depth*, and *severity filter* parameters. Examples of the latter include *string, numeric,* and *Boolean* data types. Each tool is required to register a set of standard parameters, as well as the label and data type of any custom parameters. To promote a simple and consistent user experience, custom parameters are supported but deprecated. (Deprecation is the declaration that a component should not be used in subsequent designs, but remains available to support existing designs that incorporate it.)

The design event

This event describes the standards and services required to enable a practitioner to design a study by selecting a combination of capabilities and specifying parameters that tailor the tools to the study goals. The infrastructure presents a *control panel* that lists and organizes the aggregated capabilities of all registered tools. This list is organized hierarchically, using the taxonomy of capabilities as a conceptual framework. As shown in ***Figure 2***, the user can review the available capabilities and select those that best support the research goals. In this fictional example, two tools support checking for red-green discrimination, and the user has selected the tool that has a warrant from the Royal National Institute for the Blind.[39]

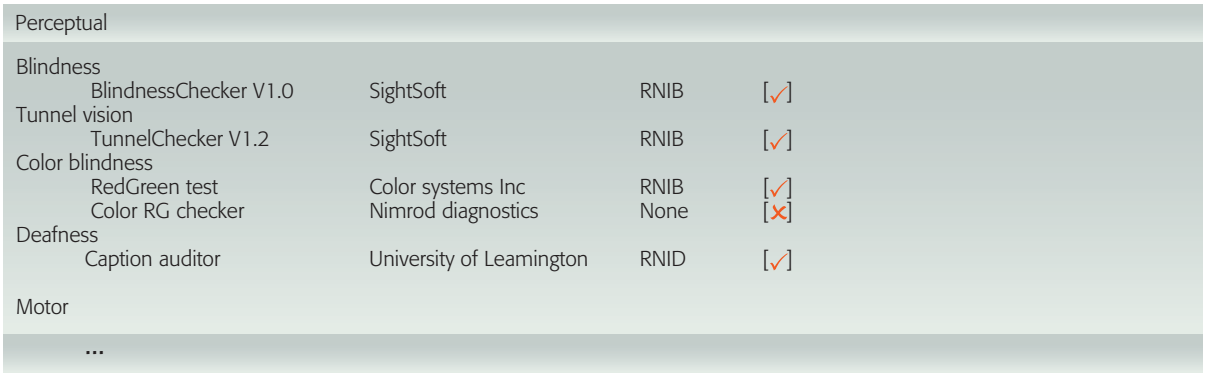

Figure 2
Design user interface

The user also needs to specify parameters at this time. When a standard parameter is used by many tools, the value need only be specified once. For example, the user might specify starting page and crawling depth in a single dialog, and the infrastructure would then pass these values to all tools that have registered a need for this information. One critical common parameter is compliance level. For example, the evaluator might use a common dialog to notify all tools to check for compliance with W3C level AA. In practice, some tools may use criteria other than the W3C ratings. In this case the parameter dialog would aggregate all criteria registered by the selected tools.

Study designs are useful intellectual property in their own right and may need to be reused or adapted to save effort and share best practice. Consequently, the user interface provides a facility to import and export designs in the form of named sets of capability selections and parameter values.

The transport event

This event describes the standards and services required to enable a practitioner to run a study. The user interface provides controls to start, pause, restart, and abandon a study. Long-running or complex studies may need to run for periods longer than a working day in situations where it is impractical or unsafe to leave a machine running unattended. Pause and restart capabilities provide a standard mechanism to take a checkpoint, suspend the test, and resume at a later convenient time.

These transport commands are passed to each of the enabled tools. Checkpoint/restart is also supported by standard services in the infrastructure for recording the state of I/O and for saving accumulated findings reported by individual tools.

The execute event

This event describes the standards and services required to enable the infrastructure to run a study designed and started by a practitioner. The infrastructure, rather than the individual tools, manages the input to the Web site to be tested. Starting at the home page specified by the designer as a standard parameter, it reads the site as a set of linked pages. For each page, it reads and parses the base content defined in HTML, PostScript**, or presentation information coded as CSS (Cascading Style Sheets), and then notifies each enabled tool that a new page is ready for testing.

Each tool then makes calls on a set of convenient input services provided by the infrastructure to request content from the page in a format suitable for the tool's specific needs. For example, a tool designed to test the reading difficulty of a page might call an infrastructure service to return the content in units of single sentences. A tool designed to check for typographical contrast might call a service to get the next distinctly styled block of text and the associated HTML and CSS markup associated with that block. Similarly, a tool interested in assessing an index of redundancy in the use of repeated hyperlinks might call a service to return the page as a DOM (document object model), whereas a tool that relies on digital-signal-processing techniques to assess color contrasts might request a rendered image of the page at a given resolution.

Additionally, the infrastructure might provide standard services to read and parse embedded content to deal with certain specialized data types, for example, time-based media encoded as AVI (Audio Video Interleave) files. A specialized tool could then register an interest in AVI files in order to check for the provision of subtitles. Further analysis is required to define an exhaustive list of the necessary input services. A plug-in approach to input services would provide an extensible solution.

Whenever a tool identifies a finding, it calls a common reporting service in the infrastructure. The service is constructed to acquire finding descriptions in a standard form that ensures completeness, consistency, aggregation, and comparison. *Table 7* lists the fields within the finding report and indicates the responsibility for each field. Note the distinction between impact and compliance. The former encodes the classes of disability affected and the associated impact on task success. The latter records noncompliance with external guidelines such as the WCAG.[4]

During execution, the infrastructure periodically requests a status update from each tool. The tool is then required to respond with a concise description of its current activity. The infrastructure combines this data with statistics on the number of pages analyzed and the count and severity of findings identified.

The analyze event

This event describes the standards and services required to analyze the findings reported by the enabled tools. At this stage, the individual tools have completed their contribution to the study, and the infrastructure need only provide services to assist the practitioner in both analyzing and reporting the results.

Typically, test tools generate inconveniently large volumes of findings. Practitioners need effective tools to tame the aggregated dataset, using quantitative and qualitative methods to identify the underlying issues of interest to stakeholders. Consequently, the architecture proposes a flexible set of analysis features to enable the practitioner to explore the data using relevant criteria to count, sort, filter, and search. For example, a practitioner might wish to review all serious problems reported with respect to a site's search page.

Alternatively, the practitioner might be interested in the relative and absolute frequency of findings cross-tabulated by disability and severity, or might need to identify all findings that impact users with motor impairment, sorted by severity. To satisfy these diverse and unpredictable goals, the user interface provides a mechanism to initially filter findings either by specifying legal values for some set of coded fields, such as impact or noncompliance, or by comparing an arbitrary search string to any text field, such as finding a description or page title. After a filtered subset has been obtained, the user can either list or cross-tabulate the results. For a list, the user would select a set of fields to display and sort. For cross-tabulation, the user would specify a numeric field to display and two encoded fields to use as row and column titles. It may also be useful to offer a facility to save, reuse, and share favorite filters, lists, and cross-tabulations.

In some cases the evaluator may wish to add a comment to explain or qualify a finding. A facility is provided to edit such manually entered comments. Evaluators may also mark a finding as deleted.

In addition to findings, some tools identify actions. For example, a tool might invite the practitioner to manually review the content of an HTML <ALT> attribute for relevance and clarity. To support such actions, the infrastructure provides a to-do list to enable the practitioner to systematically respond to each action and to update findings with additional data.

The report event

This event describes the standards and services required to export results in a form helpful to others involved in the evaluation process. Potential receivers of test results include designers, developers, and business stakeholders. It may also be useful to archive data for long-term initiatives, such as intraproject and interproject analyses of trends and regression. The needs of each audience are likely to be somewhat different. For example, a developer is primarily interested in a page-by-page enumeration of errors and recommendations, whereas a manager is more concerned with compliance issues and strategic recommendations. Additionally, different media may be needed to meet local technical strategies. Although one enterprise may wish to distribute HTML-based reports, another may prefer to work with RTF (Rich Text Format) documents.

Table 7 Fields within a report of findings

Identification Provides a unique reference Enables findings to be aggregated by tool, capability, page, or element		
ID	Generated automatically by the tool	Unique alphanumeric ID
Study name	Generated automatically by the tool from registration data using a standard parameter supplied by the user	Concise name for study to enable findings to be aggregated and compared across studies
Time stamp	Generated automatically by the tool from system clock	Date and time page loaded
Source	Generated automatically by the tool from registration data	• Capability name • Tool name, version, and author
Page	Recorded automatically by the infrastructure using the URL of the last page crawled	URL
Page title	Recorded automatically by the infrastructure using the <TITLE> tag of the last page crawled	Page title text
Element ID	Recorded automatically by the infrastructure by assigning a unique ID to each element passed in response to an input service request from a tool	Unique alphanumeric ID for an element (enables multiple findings for an element to be aggregated for a tool, and in certain cases, across multiple tools)
Element description	Recorded automatically by the infrastructure using the last input service called by the tool	A concise textual summary of an element such as a sentence or a block of HTML content (used to illustrate findings in reports and to enable developers to identify the location of the error)
Consequences Describes the effect of the finding on the user and levels of noncompliance		
Impact	Mandatory field supplied by the tool	Describes the set of disabilities affected and assesses the severity for each disability
		Values for disability are specified using a controlled vocabulary based on a broad and generic taxonomy.
		Values for severity are specified using a criteria-based scale that describes the impact of the finding on task completion
Noncompliance	Mandatory field supplied by the tool	Describes the set of guidelines within external standards that the page does not comply with.
		Specifies the level of noncompliance for each such violation
		Values are codes associated with a specific externally defined standard (for example, levels A, AA, and AAA in the W3C guidelines).

Table 7 Fields within a report of findings (continued)

Description Describes and illustrates the finding Enables findings to be communicated clearly to stakeholders and developers		
Description of finding	Mandatory field supplied by the tool	A concise textual description of the problem
Location of finding	Mandatory field supplied by the tool	A specification of the location of the problem within a page description
Illustration of finding	Optional field supplied by the tool	Concise text that reports or quotes from the erroneous code or content
Evaluator comment	Optional field entered subsequently by the evaluator during analysis.	An explanation or qualification of the finding
Recommendations Enables stakeholders and developers to form a plan		
Tactical recommendation	Optional field supplied by the tool	Describes a specific recommended change to code or content
Strategic recommendation	Optional field supplied by the tool	Describes a recommendation to change processes and policies
Actions Notifies practitioners that additional manual inspection is required to complete the finding		
Action	Optional field supplied by the tool	Instructions to perform additional expert inspection on a specific segment of code or content

Likewise, some may require CSV (Comma Separated Variable) files for conversion to spreadsheets or relational databases, while others have a strategic commitment to the benefits of XML markup.

The W3C Evaluation and Repair Tools Working Group (ERT WG) is currently developing the Evaluation and Report Language (EARL).[40] EARL is a language to express test results such as bug reports and conformance claims. EARL "enables any person, entity, or organization to state test results for any *thing* tested against any set of criteria." In a situation where a Web site is tested for conformance with the WCAG guidelines by using a variety of tools, EARL can be used to compare results among tools, identify conflicting results, and help evaluators derive a single result from a multitude of tools. Once EARL has been fully defined, the applicability of the proposed framework in supporting EARL should be addressed.

A flexible strategy is required to meet these diverse information and technology requirements. Three specific requirements are clear:

1. A mechanism is required to filter both findings and fields of interest to a specific audience.
2. Distinct output formats are needed for human readers and downstream software tools.
3. A range of established, externally specified physical formats must be supported.

The first requirement is addressed by the same facilities that the practitioner uses to filter, sort, list, and cross-tabulate findings. Whereas these features were used in analysis to select subsets of data to review and explore, they are used here to select subsets of interest to a specific audience or program. The second requirement is met by generating an XML version of the filtered data as input to a pluggable XSL (Extensible Stylesheet Language) style sheet. For example, the style sheet might transform a raw hierarchy of findings within categories into a readable and well-presented document by creating an appropriate hierarchy of headings, effective typography, and an appropriate look and feel. The final requirement involves the ability to export basic or transformed findings in a range of physical formats. Although additional research is

required to specify the entire range of required formats, an initial set might usefully include XML, XHTML, HTML, CSV, and RTF.

CONCLUSION

The architecture described in this paper addresses the identified issues of consistency and integration in automated accessibility test tools by applying pragmatic techniques from software engineering informed by a plausible and attractive business model. Three major benefits are claimed for this approach:

1. Tools are less expensive to develop. Commercial software developers are exposed to smaller investments and less risk in developing tools for specialized aspects of accessibility. Academic research teams and advocacy groups can channel finite resources to tackling small, well-bounded problems directly related to their core expertise.
2. The overall user experience is more efficient and natural. Studies are designed in terms of capabilities rather than software products. Tools are started together, executed concurrently, monitored, and can be paused and resumed as necessary. Findings are aggregated as a coherent and usefully structured dataset that can be analyzed and reported by using powerful filtering and formatting tools.
3. The eventual consumers of reports will receive documents that are appropriately structured for both understanding and action.

Further research is recommended to understand requirements in more detail, construct quantitative business models, and refine the proposed architecture in terms of a preferred implementation.

**Trademark, service mark, or registered trademark of Adobe Systems Incorporated, Eastman Kodak Company, Google, Inc., Macromedia, Inc., Massachusetts Institute of Technology, Sony Corporation, or Sun Microsystems, Inc.

CITED REFERENCES AND NOTES

1. *Universal Usability Guide*, universalusability.org, http://www.universalusability.org/.
2. B. Shneiderman, "Universal Usability," *Communications of the ACM* **43**, No. 5, 84–91 (2000).
3. *The Web: Access and Inclusion for Disabled People*, Disability Rights Commission (2004), http://www.drc-gb.org/publicationsandreports/report.asp.
4. W. Chisholm, G. Vanderheiden, and I. Jacobs, *Web Content Accessibility Guidelines 1.0*, World Wide Web Consortium (May 5, 1999), http://www.w3.org/TR/WCAG10/.
5. *Speech and Braille Output Software*, Royal National Institute of the Blind, http://www.rnib.org.uk/xpedio/groups/public/documents/PublicWebsite/public_speechbrailleoutput.hcsp.
6. IBM Home Page Reader 3.04, IBM Corporation, http://www-3.ibm.com/able/solution_offerings/hpr.html.
7. Welcome to TechDis, TechDis, http://www.techdis.ac.uk/index.php.
8. Screen Magnification Software, Royal National Institute of the Blind, http://www.rnib.org.uk/xpedio/groups/public/documents/PublicWebsite/public_screenmagnification.hcsp.
9. *Section 508: Glossary*, NASA, http://section508.nasa.gov/glossary.htm.
10. *User Engineering*, IBM Corporation, http://www-306.ibm.com/ibm/easy/eou_ext.nsf/publish/1996.
11. A. H. Maslow, *Toward a Psychology of Being, 3rd Edition*, Wiley, Hoboken, NJ (1998).
12. K. Norman, *The Psychology of Menu Selection: Designing Cognitive Control at the Human/Computer Interface*, Intellect Ltd., Bristol, UK (1991).
13. J. Nielsen, *Usability Engineering*, Morgan Kaufmann, San Francisco, CA (1994).
14. H. W. J. Rittel, "Second Generation Design Methods," Interview in *Design Methods Group 5th Anniversary Report: DMG Occasional Paper* **1**, 5–10 (1972). Reprinted in *Developments in Design Methodology*, N. Cross, Editor, Wiley, Hoboken, NJ (1984), pp. 317–327.
15. Professional discussion with blind colleague (2003).
16. *Disability Briefing December 2004*, Disability Rights Commission (2004), http://www.drc-gb.org/publicationsandreports/campaigndetails.asp?section = ddb&id = 666.
17. *Disability Discrimination Act 1995*, Disability Unit of the Department for Work and Pensions, UK, http://www.disability.gov.uk/dda/.
18. Section 508, Center for IT Accommodation (CITA), Office of Governmentwide Policy, U.S. General Services Administration, http://www.section508.gov/.
19. K. Vredenburg, *User-Centered Design: The Integrated Approach*, Prentice Hall, New York (2002).
20. J. S. Dumas and J. C. Redish, *A Practical Guide to Usability Testing*, Intellect Ltd., Bristol, UK (1999).
21. P. Englefield, *A Pragmatic Framework for Selecting Empirical or Inspection Methods to Evaluate Usability*, IBM Corporation (2003), http://www-306.ibm.com/ibm/easy/eou_ext.nsf/Publish/50?OpenDocument&../Publish/1118/$File/paper1118.pdf.
22. J. West, "The Newest AT Goes Mainstream and to the Movies: Academy Awards Party Features Innovative Access for All Courtesy of IBM," *The Assistive Technology Journal* **70** (April 2003), http://www.atnet.org/news/2003/apr03/040102.htm.
23. Testing for Accessibility, Royal National Institute of the Blind, http://www.rnib.org.uk/xpedio/groups/public/documents/publicWebsite/public_testing.hcsp#P27_2586.
24. Local Authority Websites (LAWs), UK National Projects Programme, http://www.laws-project.org.uk.
25. IBM Web Services, IBM Corporation, http://www-128.ibm.com/developerworks/web/library/w-int.html.

26. The Eclipse Project, The Eclipse Foundation, http://www.eclipse.org/eclipse/.
27. B. S. Rubin, A. R. Christ, and K. A. Bohrer, "Java and the IBM San Francisco Project," *IBM Systems Journal* **37**, No. 3, 365–371 (1998).
28. E. Rogers, *Diffusion of Innovations, 5th Edition*, Free Press, New York (2003).
29. W3C Web Accessibility Initiative (WAI), World Wide Web Consortium, http://www.w3.org/WAI/.
30. Lynx, http://lynx.browser.org/.
31. J. Knight, *Attitudes to Web Accessibility*, UsabilityNews.com (October 2003), http://www.usabilitynews.com/news/article1321.asp.
32. Inspect32 and AccExplorer32 are both part of the Microsoft Active Accessibility® 2.0 Software Development Kit. For details, see Active Accessibility 2.0 SDK Tools, Microsoft Corporation, http://www.microsoft.com/downloads/details.aspx?FamilyId=3755582A-A707-460A-BF21-1373316E13F0&displaylang=en.
33. D. J. Delorie, Lynx Viewer, http://www.delorie.com/web/lynxview.html.
34. LIFT Online, UsableNet Inc., http://www.usablenet.com/products_services/lift_online/lift_online.html.
35. LIFT Machine, UsableNet Inc., http://www.usablenet.com/products_services/lift_machine/lift_machine.html.
36. *ISO Standard 9241: Ergonomic Requirements for Office Work with Visual Display Terminals*, International Organization for Standardization, Geneva, Switzerland (1999).
37. S. Z. Nagi, "Disability Concepts Revisited: Implications for Prevention," in *Disability in America: Toward a National Agenda for Prevention*, A. M. Pope and A. R. Tarlov, Editors, National Academy Press, Washington, D.C. (1991), pp. 309–327.
38. *International Classification of Functioning, Disability and Health (ICF)*, World Health Organization, Geneva, Switzerland (2001), http://www3.who.int/icf/icftemplate.cfm?myurl=introduction.html%20&mytitle=Introduction.
39. Royal National Institute of the Blind, http://www.rnib.org.uk.
40. W. Chisholm and S. B. Palmer, *Evaluation and Report Language (EARL) 1.0*, World Wide Web Consortium (December 6, 2002), http://www.w3.org/TR/2002/WD-EARL10-20021206/.

Accepted for publication January 17, 2005.
Published online August 8, 2005.

Paul Englefield
IBM Warwick, MP5, PO Box 31, Birmingham Road, Warwick CV34 5JL, UK (paul_englefiled@uk.ibm.com). Mr. Englefield joined IBM in 1978 and currently works as a senior usability consultant in the IBM Worldwide Ease of Use (EOU) team in Warwick, UK. He leads usability engagements for internal, commercial, and government clients, runs a team developing software tools for usability practitioners, leads a corporate work group on research and evaluation methods, and teaches both usability and accessibility skills. His interests include evaluation tools, inspection methods, task analysis, and design rationale; he has published a range of articles, papers, tutorials, and training materials on these topics. He has an M.Sc. degree in human-centered technology and holds five patents related to user interface technology. He is a member of the IBM corporate UCD (User Centered Design) advisory council and the British Computer Society HCI (Human-Computer Interaction) Education and Practice group. Away from the office, Paul enjoys acting in murder mysteries, studies jazz guitar, and makes great black-cherry pancakes.

Claire Paddison
IBM Warwick, MP5, PO Box 31, Birmingham Road, Warwick CV34 5JL, UK (paddisonc@uk.ibm.com). Ms. Paddison is a usability and accessibility consultant working in IBM's Usability Competency Centre (UCC), part of the IBM Worldwide Ease of Use (EOU) Strategy and Design group in Warwick, UK. She first joined IBM as an industrial trainee at the Greenock manufacturing site in 1995. There she worked on monitor control usability and the out-of-box experience of the IBM customer. She returned to Greenock as a human factors engineer after graduating from Loughborough University with a B.Sc. degree in ergonomics. Today Claire works as a consultant advising customers on the design and evaluation of user interfaces, primarily for Web sites. She has practical experience in many areas of usability and particularly enjoys the challenge of design work. Her main area of expertise within the group is accessibility. She has published a range of articles and papers on evaluating for accessibility, acts as the accessibility focal point for her group, and also provides accessibility advice to many other groups within IBM.

Mark Tibbits
IBM Warwick, MP5, PO Box 31, Birmingham Road, Warwick CV34 5JL, UK (mark_tibbits@uk.ibm.com). Mr. Tibbits has many years of practical experience leading usability and accessibility consultancy engagements within the retail, government, automotive, banking, and insurance industries. Recent engagements include evaluating the efficiency, productivity, and accessibility of employee intranets, commerce Web sites, thin-client banking applications, and government Web sites. He also teaches usability design and evaluation to IBM product development teams and client design teams. In addition to his usability role, Mark acts as technical architect for a range of projects related to universal usability practitioner support tools.

Isha Damani
University of Warwick, Coventry CV4 7AL, UK (I.Damani@warwick.ac.uk). Miss Damani is currently in her final year at the University of Warwick majoring in computer and business studies. Her dissertation is on the subject of IT fads and foundations. Last year she participated in a work experience program with IBM Warwick, where she contributed to the studies described in this paper. She will be joining Deloitte in the fall of 2005. ■