Advances in Web Testing

CYNTRICA EATON

Department of Computer Science, Norfolk State University, 700 Park Avenue, Norfolk, Virginia 23504

ATIF M. MEMON

Department of Computer Science, University of Maryland, 4115 A. V. Williams Building, College Park, Maryland 20742

Abstract

Demand for high-quality Web applications continues to escalate as reliance on Web-based software increases and Web systems become increasingly complex. Given the importance of quality and its impact on the user experience, a significant research effort has been invested in developing tools and methodologies that facilitate effective quality assurance for Web applications. Testing, in particular, provides a critical inroad toward meeting the quality demand by enabling developers to discover failures and anomalies in applications before they are released. In this survey, we discuss advances in Web testing and begin by exploring the peculiarities of Web applications that makes evaluating their correctness a challenge and the direct translation of conventional software engineering principles impractical in some cases. We then provide an overview of research contributions in three critical aspects of Web testing: deriving adequate Web application models, defining appropriate Web testing strategies, and conducting Web portability analysis. In short, models are used to capture Web application components, their attributes, and interconnections; testing strategies use the models to generate test cases; and portability analysis enables Web developers to ensure that their applications remain correct as they are launched in highly diverse configurations.

ISSN: 0065-2458/DOI: 10.1016/S0065-2458(08)00805-X

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:31 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

1. Introduction

With a significant role in modern communication and commerce, Web applications have become critical to the global information infrastructure and, subsequently, one of the largest and most important sectors of the software industry [24, 39]. As a natural corollary, ensuring the quality of Web applications prior to release is highly important. Yet, given extreme time-to-market pressures, increasingly complex Web applications, constant shifts in user requirements, and rapidly evolving development technologies, achieving this quality is extremely difficult and presents novel challenges to software development [4, 56]. As a result, implementing high-quality Web applications using a cost-effective development process is currently one of the most challenging pursuits in software engineering; a significant research effort has been invested in developing systematic, quantifiable approaches that support Web quality assessment [4, 6, 38, 43].

Given the broad use of the word *quality* thus far, it is important to note that Web application quality is a complex, multifaceted attribute that has many dimensions including *usability* [9, 14, 35, 54], *performance* [12, 37, 51, 60], *accessibility*[1] [2, 11, 33, 46, 50, 52, 55], and *security* [25, 30, 31, 49]. While failure to assess each dimension prior to release can negatively impact the user experience, this chapter focuses on contributions to Web testing research. In particular, we explore work that applies structural and functional testing solutions to the discovery of Web failures. As a result, research devoted to usability, performance, accessibility, and security do not fall within the scope of this survey. For further clarity, because we are more interested in high-level, functional correctness, Web testing tools that verify links, validate Hypertext Markup Language (HTML) or Extensible Markup Language (XML) syntax, and perform stress testing fall outside the bounds of this discussion as well [10, 34].

While isolating faults and ensuring correct functionality is a vital process associated with any software development effort, it is widely considered tedious and time consuming even for more traditional software types with stable, well-defined, monolithic runtime environments [47, 53]; since Web applications are much more complex, the testing process can be even more involved. One overarching idea in Web testing research is to identify well-established software engineering methods that can address specific problems in Web development, adapt or modify them to account for peculiarities and complexities of Web applications, and define novel approaches when necessary [18, 45]. In this chapter, we provide more insight into how researchers are using this practice to advance the field of Web application testing by deriving solutions to three issues: the extraction of suitable test models, development of effective testing strategies, and assessment of configuration-independent quality through portability analysis. Our focus on these three particular issues aligns with the idea that effective testing of Web-based applications must include extracting models capable of representing components of the application and their interconnections, deriving and executing test cases based on those models, and ensuring that quality is preserved as Web applications are launched in diverse configurations.

We structure our discussion of Web testing research contributions in the following way: In Section 2, we take a look at how Web applications have evolved and overview characteristics that make testing them unique and challenging. In Section 3,

[1] The most widely used goal of accessibility is to ensure that Web applications accommodate the needs of physically and mentally handicapped users.

we discuss Web application models designed to capture characteristics useful for testing. In Section 4, we overview Web testing methodologies and processes. In Section 5, we discuss research in Web portability analysis where the goal is to ensure that quality does not diminish as Web applications are ported. In Section 6, we conclude.

2. Challenges of Web Testing

As use of the Web grew at a tremendous rate and the benefits of implementing high-quality Web-based systems became more apparent, the pursuit for expanded capabilities of Web applications simultaneously increased their complexity and drove the rapid evolution of Web technology. Over the years, Web infrastructure has evolved from primarily being a communication medium to a platform for elaborate Web applications that are interactive, highly functional software systems [56]. This evolution has had a notable impact on the pursuit and effective implementation of quality assurance strategies; with room for more complex interaction and increased computation, it is widely acknowledged that rigorous verification and validation approaches are necessary [41]. In the rest of this section, we discuss several factors inherent to Web development that contribute to the quality problem; in doing so, we also highlight the challenges and considerations that influence the practicality and usefulness of conventional software testing methodologies and tools.

2.1 Heterogeneity and Distribution of Components

Given current technology, Web developers are able to create software systems by integrating diverse components that are written in various programming languages and distributed across multiple server platforms; because of the ubiquitous presence of the Web, data can be transferred among completely different types of software components that reside and execute on different computers quite easily [18, 29, 39]. These factors have contributed to a significant growth in the Web services arena and sparked a keen research interest in applying semantic nets to help manage heterogeneity.[2] Since modern Web applications typically have complex, multitiered, heterogeneous architectures including Web servers, application servers, database servers, and clients acting as interpreters, testing approaches must be able to handle highly

[2] Please refer to the chapter titled ''Semantic web applied to service oriented computing'' by Fensel and Vitvar for further information.

complex architectures and account for the flow of data through the various architectural components [24, 39].

2.2 Dynamic Nature of Web Applications

There are several aspects that make Web applications highly dynamic. For one, unlike earlier Web pages that had static structure and hard-coded components, modern Web applications can react to user input, generate software components at runtime, assemble components from varied sources, and create Web pages on the fly [18, 41]. Moreover, interaction between clients and servers can change dynamically over a session depending on how users interface with a system. Finally, in Web development, application requirements routinely change because of advances in technology or in response to user demand. All combined, dynamically generated components, dynamic interaction among clients and servers, constant changes in application requirements, and continually evolving technologies make techniques that were effectively applied to simple Web applications with traditional client–server systems inadequate for testing dynamic functionality.

2.3 Unpredictable Control Flow

Variance in control flow was generally not a factor for traditional systems because flow was exclusively managed by program controllers. Since Web applications can have several entry points and users can arbitrarily navigate to previously visited Web pages by interacting with their Web browser interface, control flow in Web applications is largely unpredictable [1]. In terms of entry points, users can directly access Web pages when given the appropriate Uniform Resource Locator (URL). In cases when Web applications consist of several Web pages that are expected to be accessed in a particular order, users could find themselves at an improper starting point if they type in the URL to an intermediate page directly or they discover an intermediate page in a batch of search engine returns. To ensure proper functionality, this factor must be carefully accounted for during development to ensure that users can, in effect, find their way to the intended start page if they happen to land somewhere in the middle. Since users interact with Web applications through browsers, loose coupling of browser controls and the Web application can translate into unexpected failures and anomalies. For instance, a user can break normal control flow by refreshing a Web page or navigating to an earlier/later point in their navigation history with the help of the back and forward buttons. In either case, the execution context will have changed without notifying the program controller, possibly triggering unexpected results. To ensure that interaction with the browser

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:32 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

does not have a negative effect, browser controls and their effects must be factored in during Web application testing [19].

2.4 Significant Variation in Web Access Tools

An important challenge to Web quality assurance stems from increased diversity of client platforms. Although users traditionally explored the Web with versions of either Internet Explorer or Netscape on desktop PCs, recent trends including the emergence of Mozilla, for instance, as a popular browser alternative and shifts toward Web-enabled appliances such as televisions and personal digital assistants (PDAs) suggest that the contemporary face of Web browsing environments is continuing to evolve. While the wide variety of tools used to navigate and interact with the Web provide users with expanded flexibility in choice of access platform, it complicates Web quality assurance. In essence, wide variation translates into a wide space of potential Web client configurations and complicates the testing effort by requiring that Web developers not only ensure that the systems they have developed are correct, but that correctness persists as software is ported. Failure to evaluate Web application portability across the configuration space can result in instances when Web application components render/execute correctly in some client configurations and incorrectly in others.

2.5 Development Factors, Adjusted Quality Requirements, and Novel Constructs

The process used to develop Web applications presents a significant challenge to Web testing. Web software is often developed without a formalized process; developers generally delve directly into the implementation phase, rarely engage in requirements acquisition, and go through a very informal design phase [6, 41]. This direct, incremental development is more than likely the result of two factors: time-to-market pressure and the ability for relatively untrained developers to create and modify Web sites using tools like KompoZer,[3] Amaya,[4] and Dreamweaver[5] that support What You See Is What You Get (WYSIWYG) implementation. To accommodate these factors, testing approaches would, ideally, be automatable and incorporate easily adaptable test suites [24].

[3] http://www.kompozer.net/.
[4] http://www.w3.org/Amaya/Amaya.html.
[5] http://www.adobe.com/products/dreamweaver/.

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:32 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

Shifts in quality requirements for Web applications, in comparison to more traditional software, also impact the Web testing process. According to Wu and Offutt [56], much of the software industry has been able to succeed with relatively low-quality requirements; a combination of timely releases and marketing strategies have almost always determined whether traditional software products succeed competitively. In contrast, Web traffic is heavily influenced by software quality; since users have *point-and-click* access to competitors, they can very easily take their business elsewhere. As a result, the goals of the development process must be reprioritized since producers only see a return on their investment if their Web sites meet consumer demand.

Finally, Web applications incorporate a host of novel constructs; integrating practices for adequate assessment during testing is key. For one, modern Web applications often have interface components that are completely hidden in that they do not correspond to any visible input elements on a Web page [26]. As a result, it is a important to support analysis for Web applications that take hidden elements into account; incomplete information about interfaces can limit the effectiveness of testing and preclude testers from exercising parts of an application that are accessible only through unidentified interfaces [1, 26].

2.6 Summary

In summary, Web applications can be described as heterogeneous, distributed systems that are highly dynamic with unpredictable control flow. Since Web applications have high-quality demands, are expected to run on a wide variety of client configurations, and incorporate novel constructs, it is important that testing approaches adequately address these factors as they apply. In the sections that follow, we take a look at how researchers are meeting these challenges in defining Web application models, Web testing strategies, and Web portability analysis approaches.

3. Web Application Models

In the field of software engineering, models are often used to aid developers in analysis. In general, models help to capture software features relevant to testing by abstracting components, their attributes, and interconnections; models can represent varying degrees of granularity depending on the features salient to the testing approach. This section provides an overview of various Web application modeling techniques and establishes a context for the testing strategies discussed in Section 4.

In particular, we discuss various approaches including Markov models and statecharts, object-oriented models, and regular expressions. In the next section, we look at how these models are used to derive test cases and evaluate the functional and structural correctness of Web applications.

It is important to note that, in their work, Alalfi, Cordy, and Dean [1] surveyed close to two dozen Web application models used to support Web testing and provided a comprehensive discussion of the various techniques used to model navigation, content, or behavior of Web applications. There is only a slight overlap in the models discussed in Ref. [1] and in this chapter.

3.1 Markov Models and Statecharts

Kallepalli and Tian [32] proposed unified Markov models (UMMs) for testing Web applications. In essence, UMMs are variants of Markov models that are defined as a set of hierarchical Markov chains where states are operational units (Web files), edges between states correspond with hypertext links and indicate a possible transition (navigation), and usage probabilities indicate the likelihood of a transition. The UMM shown in Fig. 1 represents a simple Web application that is comprised of a homepage (the intended start page) with links to frequently asked questions (faq) and a registration page. From the faq page, the user can either navigate back to the homepage or leave the Web application altogether; from the registration page, users can either return to the homepage, go to a confirmation page, or quit the Web application. The probability of making transitions between Web pages is listed

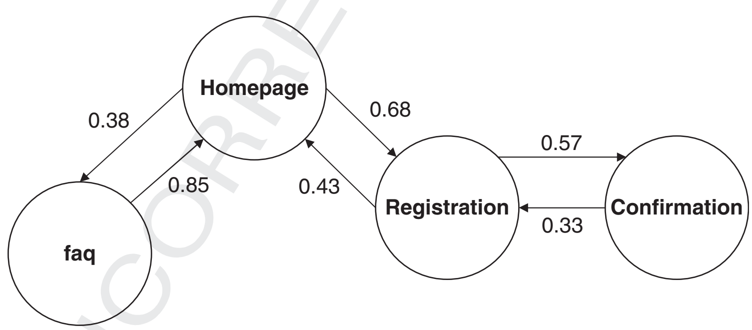

FIG. 1. Unified Markov model (UMM) example for a simple Web application. States represent individual Web pages, edges correspond to hyperlinks, and the probabilities shown represent the likelihood that a user will select the corresponding hyperlink from a certain page.

alongside the edges; as an example, users navigate from the homepage to the faq 38% of the time. The underlying idea is to have the UMM represent execution flow, information flow, and probabilistic usage information. States, edges, and usage probabilities are each recovered from Web logs that maintain a record of user interaction with the system including usage frequency and failure reports; since Web logs are quite common and routinely maintained on the server side, this approach incurs low overhead. The models extracted are eventually used to support statistical testing.

There is quite a bit of similarity between the work of Kallepalli and Tian [32] and that of Sant *et al*. [48]; both use Markov models (or some variation thereof) to represent Web applications, generate the model based on logged user data, and use the model as a basis for testing. The major difference between the two is that Sant *et al*. experiment with using varying degrees of history to estimate whether users will visit a given Web page during a session. In particular, they look at *unigram* models where page visitation is considered independent of previous actions, *bigram* models where the previous Web page visited has an impact, and *trigram* models where the previous two pages help to define the probability that users will visit a given page.

Statecharts generally model reactive systems as a series of states, transitions, events, conditions, and their interrelations. Di Lucca and Penta [19] proposed a Web application model based on statecharts that can be used to analyze how interaction with the Web browser interface affects Web application correctness. As mentioned in Section 2, users can disrupt normal control flow and cause anomalous behavior of Web-based system by refreshing a Web page or navigating to recently visited Web pages using either the forward or back buttons. As a result, it is important to detect problems that may unintentionally arise from user interaction with the Web browser interface. Di Lucca and Penta [19] presented the following model: the browser is characterized by the Web page displayed, by the state of its buttons (*enabled* or *disabled*), and the history of Web pages visited using the browser buttons. Each of these features is captured in a statechart, where each state is defined by the page displayed and by the state of the buttons while the user actions on page links or browser buttons determine the state transactions. Consider the statechart shown in Fig. 2; in this example, the user starts at a search engine, gets a list for search results from a query, and follows a link of interest. Each of the states is labeled with a brief description of the page loaded in the browser (i.e., search engine) and the state of the back and forward buttons. As an example, if the back button is disabled and the forward button is enabled, the corresponding label would be *BDFE* where *B* corresponds with back, *D* indicates that the button is disabled, *F* corresponds with forward, and *E* indicates that the button is disabled.

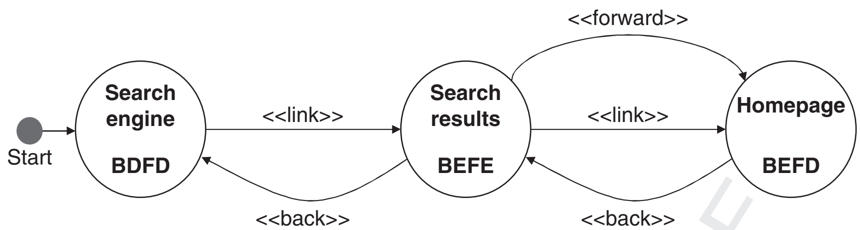

FIG. 2. Statechart example for a simple user session. In this example, the user starts at a search engine, submits a query, and activates a link to retrieve search results. From there, a link is activated to reach the homepage of a particular search return. Note, once the user reaches the homepage, they can only return to the previous page, search results, since there are no links to another page; as a result the forward button is disabled and the back button is enabled.

3.2 Object-Oriented Models

Several researchers have explored the use of object-oriented models for Web applications. This is largely because components and attributes of Web applications can be easily and accurately represented using object-oriented models and using such models facilities application of pre-existing object-oriented software testing techniques [57]. In this section, we explore object-oriented support for Web application modeling. In general, with object-oriented approaches, the central entity in a Web site is the Web page; a Web page contains the information to be displayed to the user and the navigation links toward other pages. It also includes components that facilitate organization (i.e., frames) and interaction (i.e., forms). Web pages can be static or dynamic; while the content of a static page is fixed, the content of a dynamic page is computed at runtime by a server and may depend on input provided by the user through input fields. Subclasses of the Web page model are generally defined to capture differences between the two.

One of the earlier papers defining an object-oriented approach to Web application modeling was written by Coda *et al*. [16] and provided an overview of WOOM (Web object-oriented model). Defined as a modeling framework that could be used to support Web site implementation, WOOM instances were designed to interface between the underlying concept for a Web site and its actual implementation. WOOM uses resources, elements, sites, server, links, and transformers to define Web sites. Liu *et al*. [36] introduced an object-oriented model, called the Web Application Test Model (WATM), that was designed to support data flow testing for Web applications. Liu *et al*. use static analysis of source files to create a model that represents Web applications as a set of interactive software components. Components include client pages, server pages, or program modules; attributes can be program variables or widgets and operators are defined as functions written in scripting or programming languages. Xu and Xu [57] defined an object-oriented model with three levels: the object model, the interactive relation model, and the architecture model. The object model captures attributes of and possible actions on objects (Web page components); the interactive model captures relationships between objects (how Web components influence and connect with each other); and the architecture model provides an overview of the Web application as a whole. These three levels were designed to, respectively, support unit, integration, and system testing.

A significant research effort in the area of object-oriented Web application models has been invested in extending and applying the Unified Modeling Language (UML), a family of languages primarily used in modeling and specification of more traditional object-oriented systems [1]. Very early on, Conallen [17] introduced extensions to UML, namely a new set of class and association UML stereotypes, that could support the capture of Web application-specific elements; the idea behind [17] was to provide a common way for application designers to express the entirety of their applications design with UML. Note both WOOM and the work by Conallen were motivated by design-based goals as opposed to testing; they are primarily mentioned here because of their novelty in this area when they were introduced.

Ricca and Tonella [42–45] developed a UML metamodel for high-level representation of Web applications which supports evaluation of static site structure and can be used to semiautomatically generate test cases. The analysis model primarily captures navigation and interaction capabilities [42] and it is derived from artifacts used by a Web server such as Common Gateway Interface (CGI) scripts as well as information manually provided by developers [42]. When performing testing, Ricca and Tonella reinterpret the UML model into a graph by associating objects with nodes and associations with edges. This enables traditional analyses that use graphs as a basis, such as traversal algorithms, to be applied; simple analysis can detect unreachable pages and support flow analysis to detect data dependences.

Di Lucca *et al*. also base their Web application model on UML. In Ref. [20], the authors present a tool that supports construction of UML diagrams for Web applications that lack design documents; they use UML to depict several aspects of a Web application including its structure and static/dynamic behavior at different abstraction levels. The UML diagram is generated by a tool that analyzes the source code of the application, extracts and abstracts relevant information form it, and populates a repository with the recovered information.

Bellettini *et al*. [5] discuss WebUML, a tool that generates UML models using static analysis to extract the navigational structure and dynamic analysis to recover behavior-related information about the application. In particular, WebUML constructs class and state diagrams through static source code analysis and dynamic

Web server interaction. Class diagrams represent components of a Web application including forms, frames, applets, and input fields; state diagram models are used to model entities such as active documents, which couple HTML with scripting code, and capture function call flow and navigation to other entities. It is important to note that dynamic analysis is performed by generating a set of server-side script mutants and using them in a navigation simulation; the Web pages that result from the previous step are then analyzed using static source code techniques [5].

3.3 Regular Expressions

Two lines of research incorporate regular expression notation in modeling Web applications; the idea in each is to capture structural information (i.e., arrangement of text, widgets, etc.) and possible arrangement of dynamic content (i.e., two widgets as opposed to one when the user provides a given input). Wu and Offutt [56] model individual Web pages as regular expressions to represent the static arrangement of Java servlet-based software and the dynamic sections that can vary from instance to instance. In their approach, the overall Web page P is comprised on various elements p_n. Dynamic sections are modeled as the basic elements that can be generated and standard regular expression notation is used to concisely model conditions on the appearance of each in the final HTML file. As an example, consider that $P \rightarrow p_1 \cdot (p_2 \mid p_3)^* \cdot p_4$ indicates that p_1 and p_4 will always be at the beginning and end of the corresponding HTML file; this captures the static arrangement of Web page elements. Meanwhile, either p_2 or p_3 can occur 0 or more times in the resulting page; this of course represents the dynamic nature of the page. This is a basic example of their overall approach but it captures the spirit of their work quite nicely.

In the second line of research, Stone and Dhiensa [54] present a generalized output expression that represents every possible output. While the spirit and basic motivation behind [54, 56] are the same, the former includes more advanced notation that allows other interaction factors (i.e., the affect of browser interaction on the state of dynamic elements) to be represented and the latter uses metatags instead of standard regular expression notation.

4. Web Test Case Generation

One of the basic goals of Web testing is fault discovery; characterizing the faults that affect Web applications, developing methodologies for deriving test cases, and establishing effective testing strategies have been active research areas.

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:35 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

For instance, in their work, Ricca and Tonella [44] derived a Web application fault classification model by analyzing publicly available fault reports for Web applications. While some faults included in the model occur in conventional software, others are more specific to Web applications and arise from their peculiarities. The faults in the model include authentication issues, hyperlink problems, crossbrowser compatibility (which we call portability; see Section 5), Web page structure errors, cookie/value setting issues, and incorrect protocols. In this section, we look at how the Web application models introduced in Section 3 are used to generate test cases and provide a basis for testing techniques that support fault discovery in Web applications.

4.1 Markov Models and Statecharts

Kallepalli and Tian [32] use UMMs to model usage patterns in Web applications; in particular, their model captures the likelihood that users transition from one page to the next and can be used to determine the probability of a given path from an arbitrary source state (Web page) to a sink state. To support statistical testing, Kallepalli and Tian suggest setting a probability threshold and exercising each navigation path with a higher likelihood; this approach focuses testing efforts on the most likely usage scenarios. Hao and Mendes [27] replicated this work and show that UMMs are effective in statistical testing. They extended the work of Kallepalli and Tian to account for the existence of various entry nodes, or start Web pages, in the course of a usage session. As noted in Section 2, users can start at various places in Web applications. To account for this factor Hao and Mendes use various UMMs to model a Web site, each with a different entry node; for clarity, Kallepalli and Tian only use one UMM. Similarly, Sant *et al.* [48] discuss generating test cases from random walks through Markov models.

Di Lucca and Penta [19] designed a Web application model to help developers evaluate how interaction with Web browser buttons could adversely affect system functionality. Recall, the model they developed is a statechart in which each state is defined by the Web page displayed in the browser and the status of the forward, backward, and refresh buttons; transitions between states occur when a new Web page is loaded (through a link) or either of the three buttons is activated. Di Lucca and Penta expect this approach to be integrated with other testing strategies; once a set of source-to-sink test case paths have been generated using some other approach, the idea is to create a statechart that corresponds to that sequence and include the effect of activating the forward, backward, and refresh buttons as states. The next step would be to define the coverage criteria that must be satisfied and generate a test suite that meets the given criteria. In this work, Di Lucca and Penta primarily outline

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:35 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

a model to complement existing techniques; the testing approach ultimately applied is left open.

4.2 Object-Oriented Models

Liu *et al.* [36] extend data flow testing techniques to HTML and XML to ensure that Web application data is stored, computed, and used properly. In data flow testing, program execution paths are selected based on definition–use[6] chains of variables. Since script variables and document widgets store the variables in Web applications, they are a primary target for this approach. In particular, script variables, widgets, and the Web pages that contain them are each considered objects with attributes; relationships among objects are used to generate test cases that monitor the flow of data. To accommodate the various data dependencies, Liu *et al.* define intraobject testing where test paths are selected for the variables that have definition–use chains within the object, interobject testing, where test paths are selected for variables that have def-use chains across objects, and interclient testing, where tests are derived to evaluate data interactions among clients. To test Web applications, definition–use chains need to be extended to HTML and XML documents and to cross HTTP client/server boundaries.

Ricca and Tonella [42–45] essentially use a graph-based version of their UML model to generate test cases and perform Web application testing. The tool they developed for testing is called TestWeb and the tool they use for model extraction is called ReWeb. In their approach, Web application test cases are represented as a sequence of URLs (to correspond with the Web pages in a navigation path) and values for form inputs when necessary. To derive test cases, TestWeb uses the Web application model extracted by ReWeb to generate a set of navigation paths based on some coverage criteria; testers are then responsible for manually providing values for form inputs. Once the paths are defined and values have been provided, TestWeb then automates test case execution; testers must then evaluate the results to distinguish passing test cases from failing ones. One limiting factor in this approach is the need for testers to provide form input values. In response to this issue, Elbaum *et al.* [24] apply the same basic technique as Ricca and Tonella but they further automate this approach by using data captured in actual user sessions to supply form inputs; the goal of this work is to minimize the need for tester intervention.

Finally, Bellettini *et al.* [6] introduce a semiautomatic technique for test case definition that uses a UML model at its base. Much like Kallepalli and Tian [32],

[6] Note, definition–use chains correspond to the definition of a variable v in a given program and all reachable uses of v that occur prior to any redefinition.

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:35 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

Bellettini *et al*. use knowledge of previous user interactions to determine the most exercised navigation paths and focus the testing effort on them. The tool they have developed to implement this technique, TestUML, is a testing suite that uses generated models to define test cases, coverage testing criteria, and also reliability analysis.

4.3 Regular Expressions

Regular expressions have been used to model Web applications because, using the notation, a compact representation of structure, variation, and iteration of HTML files can be expressed in a generalized way. Wu and Offutt [56] use regular expressions to model Web applications and characterize test case execution as a sequence of interactions between client and servers that begin at a source Web page and uses composition and transition rules to reach the sink. Variation in transition rules can lead to different navigation paths and each of those paths can be used as a test case.

Stone and Dhiensa [54] also use regular expressions to model Web applications and, like Wu and Offutt, are motivated by a need to evaluate dynamic Web pages to find errors. The goal of this work, in particular, is to minimize the possibility for code support errors in all possible output from a script. The basic idea of their approach is to compare the expected structure of script statements with the output actually produced. The authors suggest that only a slight extension to currently existing tools is needed to ensure they can accept and validate the more generalized model.

5. Portability Analysis

Though the process of detecting and correcting faults in an implemented software system is inherently difficult, software quality assurance becomes increasingly complex when faults only surface in precise configurations [28]. In such cases, the number, nature, and interconnection of constituent parts that define the configuration[7] can significantly impact software quality. To adequately reduce the number of faults in the delivered product, developers must evaluate the overall correctness of the implementation in addition to how that correctness is affected by variation in configurations. We refer to the process of detecting and diagnosing faults that are only triggered in precise configurations as *portability analysis*. Without an efficient, thorough technique for assessing software portability, quality could degrade as software is ported and configuration faults, or faults that are only activated in

[7] http://www.chambers.com.au/glossary/configur.htm.

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:36 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

specific configurations, have the potential to remain latent until they are encountered by users in the field.

While configuration faults affect portability for a wide range of software types, they are a particular challenge in Web application development. Given that there are several different browsers,[8] each with different versions,[9] a number of operating systems on which to run them,[10] and dozens of settings,[11] users have expanded flexibility in Web access options and the client configurations used to explore the Web are highly varied. Though this expanded variation and flexibility allows for more customized Web user experiences, subsequent differences across configurations present a serious challenge for Web developers to ensure universal quality.

Ideally, Web applications would behave and execute uniformly across heterogeneous client configurations; in such a situation, quality assurance could effectively be carried out on one client configuration and the results extrapolated for the entire set. Yet, in practice, the makeup of the client configuration has a significant impact on Web application execution (Fig. 3). Since Web applications are expected to enable crossplatform access to resources for the large, diverse user community, it is important to evaluate how well a given system meets that demand [32].

In the previous sections, most of the discussion centered on testing Web applications to ensure functional and structural correctness. In this section, we discuss various Web application portability analyses which include launching Web applications in varied configurations, looking for unsupported HTML in source code, and attempting to transform code into a form that is supported. In particular, we outline existing approaches along with their limitations and briefly discuss tools that implement them.

5.1 Manual and Automated Execution-Based Approach

Execution-based approaches to Web portability analysis primarily involve launching Web applications in target configurations and qualitatively comparing expected and observed results to verify correctness. In practical terms, this means that Web applications must be physically loaded to perform execution-based quality assurance. In the brute-force application of this approach, Web application

[8] For example, Microsoft Internet Explorer (IE), Netscape, AOL Browser, Opera, Mozilla, Safari for Mac OS X, Konqueror for Linux, Amaya, Lynx, Camino, Java-based browsers, WebTV.

[9] For example, IE 4.0, IE 5.0, IE 6.0.

[10] For example, Windows, Power Macintosh.

[11] For example, browser view, security options, script enabling/disabling.

FIG. 3. When rendered in (A) Internet Explorer 6.0 and (B) Netscape 4.8, both on Windows XP, the Scrabble Homepage is significantly different.

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:37 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

deployment and analysis are both carried out manually. Though exhaustive coverage of the configuration space would allow thorough portability analysis, physical access to each possible browsing environment is extremely difficult and nearly impossible; as a result, there is a notable conflict between the need to test each potential client configuration and the constraints imposed by limited development resources. Testing on all necessary combinations of hardware, operating systems, browsers, connection settings, etc., normally requires labor-intensive setups on many dedicated machines making it extremely difficult for a test team working with limited equipment to replicate certain configuration faults. Even with access to each possible configuration, the time and effort required to effectively assess Web pages using this strategy can also impede the depth of the Web application evaluated. Because this approach can be weakened by client configuration availability and limited time, this technique is highly ineffective and impractical for Web developers interested in establishing portability across a vast, richly defined configuration space.

While the brute-force strategy evaluates Web application portability postimplementation, Berghel [7, 8] presented a manual execution-based approach designed for preimplementation use. The basic idea outlined in Refs. [7, 8] is to launch a suite of test Web pages, called *Web Test Patterns*,[12] and use the results to gauge the level of HTML support in varied configurations. This approach allows Web developers to derive a cognitive model of HTML support criteria across various configurations; they could then use this model to drive decisions regarding which HTML tags to include in an implementation during code synthesis. Much like the brute-force strategy, the effectiveness of this approach is mainly restricted by resource limitations. In addition, this strain of Berghel's approach only allows users to develop a mental model of tag support criteria; effective application of this model can be severely flawed in practice given the expansive set of HTML tags that can be included in source code and the intricacy of support criteria. Retaining this information and attempting to use this strategy effectively is clearly time-, cognition-, and resource-intensive.

To minimize the effort and, ultimately, the cost of analysis using execution-based approaches, researchers have explored collapsing the space of test configurations through combinatorial testing approaches. In particular, Xu *et al.* [58, 59] propose applying single-factor and pairwise coverage criteria to systematically reduce the space of distinct configurations evaluated during quality assurance. This process applies sampling heuristics to define the minimal set of client configurations that must be assessed to establish confidence in the entire configuration space. While this approach can make subsequent analysis more cost-effective in terms of resources

[12] Each Web test pattern in the suite incorporates several HTML tags and descriptions of the impact they would have if processed correctly.

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:37 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

and effort, it can also create false confidence in analysis results when the set of test configurations does not accurately represent the entire space.

Commercial tools, like Browser Photo[13] and BrowserShots,[14] that are designed to make execution-based approaches more cost effective mainly automate the launch of Web applications in varied configurations to mitigate the necessity for in-house access to configurations during quality assurance. Such tools work on the behalf of Web developers by launching applications in a set of target configurations and capturing a screenshot of the rendered result; developers assess Web application correctness by manually examining the returned screenshots and relying on visual cues (i.e., misrendered pages) to discover errors. Once errors are detected, the developer must employ additional methods, such as manually examining source code, to identify fault causes. The main flaws of this approach stem from the fact that fault detection efficacy is generally constrained by the dimensions of the screen capture and the extent to which the set of client configurations used during analysis accurately represent the entire configuration space. In other words, since the result of this approach only yields visual evidence of an error, faults triggered by user action or those that fall out of the range of the screenshot will remain undetected since a single snapshot cannot capture such defects. Also, since there is no indication as to why the error occurred, it is *nondiagnostic*; identifying factors that contribute to the anomaly requires more work and effort. In addition, if the space of configurations is not adequately inclusive, critical faults could remain undetected.

In general, execution-based approaches are deficient because of limited configuration coverage, lack of diagnostic ability, limited applicability of results or some combination of these factors. As a result, practical implementation of execution-based strategies generally involves configuration sampling. Such issues give rise to an incomplete, resource-intensive analysis of the Web application that does not provide an adequate basis for establishing confidence in Web application portability.

5.2 Lookup-Based Approach

Lookup-based approaches, like Doctor HTML[15] and Bobby,[16] detect configuration faults by maintaining an account of unsupported HTML tags in a predefined subset of Web configurations and essentially scanning source code for them.

[13] http://www.netmechanic.com/browser-index.htm.
[14] ttp://browsershots.org/.
[15] http://www2.imagiware.com/RxHTML/.
[16] http://www.watchfire.com/products/webxm/bobby.aspx.

Results of analysis are returned as a list of the unsupported tags found in the given Web application source code and the configurations with support violations.

One problem of this approach is captured nicely by Fig. 3. In this example, quality is clearly diminished for Netscape users of the Scrabble Web site, however, Doctor HTML did not include this particular support violation, namely lack of support for the HTML tag <div style=background-image:url(. . .)>, in the analysis report. This factor drives home the point that this type of analysis will only be as thorough as the knowledge of configuration support criteria utilized. In instances when incomplete or inaccurate support criteria is used, configuration faults will continue to remain latent after analysis. Since the tool approach is proprietary, it is unclear whether this oversight occurred because the given HTML tag was missing from the checklist.

In our own work [21–23], we define an advanced framework that incorporates aspects of both the lookup-based and execution-based approaches. In particular, we have defined an automated, model-based framework that uses static analysis to detect and diagnose Web configuration faults. Our approach overcomes the limitations of current techniques by enabling efficient portability analysis across the vast array of client environments. The basic idea behind this approach is that source code fragments [i.e., HTML tags and Cascading Style Sheet (CSS) rules] embedded in Web application source code adversely impact portability of Web applications when they are unsupported in target client configurations. Without proper support, the source code is either processed incorrectly or ignored and the aesthetic or functional properties associated with the code may be lost resulting in configuration faults. Our approach is to model source code support in various configurations and perform portability analysis by checking for support violations in source code inclusion. In the effort to fully exploit this approach, improve practicality, and maximize fault detection efficiency, manual and automated approaches to client support knowledge acquisition have been implemented, variations of Web application and support criteria models have been investigated, and visualization of configuration fault detection results has been explored. To optimize the automated acquisition of client support knowledge, alternate machine learning strategies have been empirically investigated and provisions for capturing tag/rule interaction have been integrated into the process.

Figure 4 provides a high-level overview of four processes implemented in our framework. In particular, updateKB() is used to acquire knowledge of code support; processURL() and query() are key in portability analysis; and generateReport() is mainly responsible for presenting analysis results. To initiate analysis, Web developers submit the URL associated with the homepage of a Web application to the *Oracle* and processURL() activates a Web crawler that retrieves Web application source code and forwards it to query(). Next, query() analyzes the source code to detect support violations. Any violations discovered are presented to the Web developer by way of generateReport(). It is important to note that the integrity of the report generated is largely a factor of how comprehensive the knowledge base is;

Comp. by: VElamathiProof0000885445 Date:12/1/09 Time:07:36:39 Stage:First Proof File Path://pchns1301/WOMAT/Production/PRODENV/0000000001/0000012120/0000000016/0000885445.3D Proof by: QC by: ProjectAcronym:BS:ADCOM Volume:75005

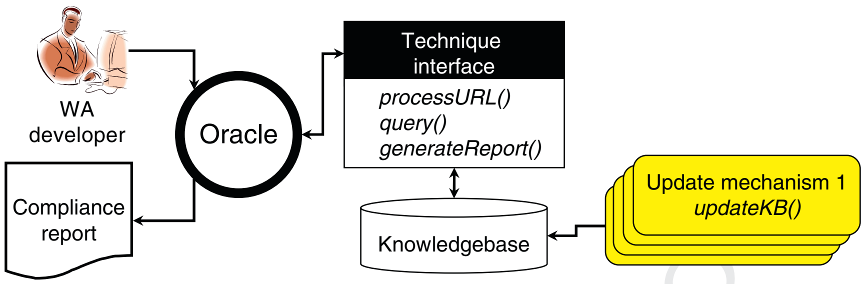

FIG. 4. High-level overview of a framework for detecting configuration-specific faults in Web applications.

subsequently, updateKB() allows both automated updates using machine learning methods and manual updates in which Web developers import support rules they, more than likely, know from experience. The underlying goal of automated, or machine learning-based, knowledge updates is to compare the source code of Web application components, namely Web pages, that render/execute properly in a given configuration with those that do not to discover possible support violations. If, for instance, a given tag consistently appears in Web pages that do not execute properly yet never in a correct Web page, it is expected to be unsupported.

In comparison to execution-based techniques, our approach bypasses the need to launch Web applications and applies a static, model-based analysis. This enables more efficient fault detection and diagnosis by reducing the need for configuration access and simultaneously reducing the threat of inaccurate equivalence assumptions. In terms of lookup-based approaches, our work uses a more inclusive model of both HTML and CSS crossconfiguration support during analysis; integrates diverse knowledge acquisition strategies to build an accurate, thorough model of support knowledge; and incorporates an extensible knowledge base model that allows support criteria to continually evolve.

5.3 Source Code Transformation Approach

Though Chen and Shen [13] do not specifically focus on Web configuration fault detection, correcting Web portability threats is a key aspect of their work and is highly applicable to the domain of Web portability analysis. In their research, Chen and Shen base their approach on the assumption that Web source code standards, as defined by The World Wide Web Consortium (W3C),[17] provide the most effective

[17] http://www.w3.org/.

basis for developing Web applications that are portable. The crux of their technique is to transform the source code of a Web application into a standardized form in which all nonstandard code fragments have been eliminated from the source yet the appearance of the original implementation is preserved. One problem with this approach stems from the fact that, as noted by Phillips [40], even if browsers fully comply with published standards, source code may still be processed differently since standards do not address every detail of implementation; in addition, there are instances in which browsers claim to be standards-compliant yet some HTML tags deemed standard by the W3C are unsupported or supported improperly [15]. In some instances, Web developers only get acquainted with the parts of the standards that work in most browsers through experience [40]; subsequently, developers may still have to employ a variant of the execution-based approach to assess source code support in client configurations.

Artail and Raydan [3] address the problem of enabling Web applications designed and tested on desktops to render properly on small-screen devices such as PDAs. At the root of the problem, mobile devices have constraints in resources and processing capabilities that make them unable to launch the vast majority of Web pages developed for desktop computers properly; Artail and Raydan describe a method for automatically re-authoring source code so that Web applications can render on a smaller screen and maintain the overall integrity of the original structure. The crux of the approach probes HTTP request headers to detect when clients are small-screen devices, to obtain dimensions of the screen size, and to use that information as a guide to transform the original source into a more compatible version while preserving the structural format of the Web page. Artail and Raydan use heuristics to reduce the size of page elements, resize images, hide text, and transform tables into text. While screen dimension constraints provide significant motivation for this work, it is important to note that there are also constraints on computing power and other resources as well; subsequently, Artail and Raydan retrieve the original source code all at once and incorporate Javascript code to display/hide parts of the Web page without having to revisit the server. This ultimately reduces user wait time considerably, saves battery power, and minimizes wireless network traffic.

6. Conclusion

Web development presents a myriad of unique challenges and requires adapted or newly developed techniques for various stages of the process. As one of the most widely used class of software to date with continually evolving design technologies, increased expectation of correctness from the user community, and short time-to-market pressures, a need for more rigorous testing approaches that can effectively reveal faults is key. Researchers are currently working to address testing problems for Web applications and to propose effective solutions.

This chapter explored research contributions that would help to enable cost efficient, effective testing of Web-based applications. In particular, we looked at the challenges involved in Web testing and discussed Web application models used, various Web testing strategies, and Web portability analysis approaches. In terms of models, we looked at variant uses of Markov models and statecharts, object-oriented models, and regular expressions. We used the discussion of those models as a basis for exploring various Web testing approaches. We then provided an overview of research efforts aimed at developing approaches for effective discovery of Web configuration faults; the goal of work in this area is to fulfill the challenge of the Web to provide configuration-independent quality to users.

As with testing research directed toward more conventional software systems, the quest to achieving quality is rather elusive and continually evolving. We expect future work in Web application testing to build upon the ideas expressed in this chapter and to become increasingly important as the Web continues to grow and evolve.

References

[1] Alalfi M. H., Cordy J. R., and Dean T. R., 2007. A survey of analysis models and methods in website verification and testing. In *ICWE, Volume 4607 of Lecture Notes in Computer Science*, L. Baresi, P. Fraternali, and G.-J. Houben, eds. Springer, Berlin. ISBN 978-3-540-73596-0.

[2] Amsler D. A., 2003. Establishing standards for usable and accessible user services web sites. In *SIGUCCS'03: Proceedings of the 31st Annual ACM SIGUCCS Conference on User Services*. ACM Press, New York, NY.

[3] Artail H. A., and Raydan M., 2005. Device-aware desktop web page transformation for rendering on handhelds. *Personal and Ubiquitous Computing*, **9:** 368–380. ISSN 1617-4909.

[4] Atkinson C., Bunse C., Grosz H.-G., and Kühne T., 2002. Towards a general component model for web-based applications. *Annals of Software Engineering*, **13:** 35–69. ISSN 1022-7091.

[5] Bellettini C., Marchetto A., and Trentini A., 2004. WebUML: Reverse engineering of web applications. In *SAC'04: Proceedings of the 2004 ACM Symposium on Applied Computing*. ACM Press, New York, NY. ISBN 1-58113-812-1.

[6] Bellettini C., Marchetto A., and Trentini A., 2005. TestUML: User-metrics driven web applications testing. In *SAC'05: Proceedings of the 2005 ACM Symposium on Applied Computing*. ACM Press, New York, NY. ISBN 1-58113-964-0.

[7] Berghel H., 1995. Using the www test pattern to check HTML compliance. *Computer*, **28:** 63–65.

[8] Berghel H., 1996. HTML compliance and the return of the test pattern. *Communications of the ACM*, **39:** 19–22.

[9] Bevan N., Barnum C., Cockton G., Nielsen J., Spool J., and Wixon D., 2003. The ''magic number 5'': Is it enough for web testing? In *CHI'03: CHI'03 Extended Abstracts on Human Factors in Computing Systems*. ACM Press, New York, NY. ISBN 1-58113-637-4.

[10] Braband C., Møller A., and Schwartzbach M., 2001. Static validation of dynamically generated HTML. In *PASTE'01: Proceedings of the 2001 ACM SIGPLAN–SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*. ACM Press, New York, NY. ISBN 1-58113-413-4.

[11] Brajnik G., 2004. Using automatic tools in accessibility and usability assurance processes. In *LNCS Proceedings of the 8th ERCIM Workshop on User Interfaces for All*. Springer, Berlin.

[12] Cai Y., Grundy J., and Hosking J., 2007. Synthesizing client load models for performance engineering via web crawling. In *ASE'07: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*. ACM Press, New York, NY. ISBN 978-1-59593-882-4.

[13] Chen B., and Shen V. Y., 2006. Transforming web pages to become standard-compliant through reverse engineering. In *W4A: Proceedings of the 2006 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*. ACM Press, New York, NY. ISBN 1-59593-281-X.

[14] Chi E. H., Rosien A., Supattanasiri G., Williams A., Royer C., Chow C., Robles E., Dalal B., Chen J., and Cousins S., 2003. The bloodhound project: Automating discovery of web usability issues using the Infoscent™ simulator. In *CHI'03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, New York, NY. ISBN 1-58113-630-7.

[15] Clark J., 2000. The glorious peoples myth of standards compliance, http://joeclark.org/glorious.html.

[16] Coda F., Ghezzi C., Vigna G., and Garzotto F., 1998. Towards a software engineering approach to web site development. In *IWSSD'98: Proceedings of the 9th International Workshop on Software Specification and Design*. IEEE Computer Society, Washington, DC.

[17] Conallen J., 1999. Modeling web application architectures with UML. *Communications of the ACM*, **42:** 63–70. ISSN 0001-0782.

[18] Di Lucca G. A., and Fasolino A. R., 2006. Testing web-based applications: The state of the art and future trends. *Information and Software Technology*, **48:** 1172–1186. ISSN 0950-5849.

[19] Di Lucca G. A., and Penta M. D., 2003. Considering browser interaction in web application testing. In *Proceedings of the 5th International Workshop on Web Site Evolution* IEEE Computer Society, Washington, DC. ISBN 0-7695-2016-2.

[20] Di Lucca G. A., Fasolino A. R., Pace F., Tramontana P., and de Carlini U., 2002. Ware: A tool for the reverse engineering of web applications. In *CSMR'02: Proceedings of the 6th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, Washington, DC.

[21] Eaton C., and Memon A. M., 2004. Evaluating web page reliability across varied browsing environments. In *Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering (ISSRE'04)*, Saint-Malo, Bretagne, France.

[22] Eaton C., and Memon A. M., 2004. Improving browsing environment compliance evaluations for websites. In *Proceedings of the International Workshop on Web Quality (WQ 2004)*.

[23] Eaton C., and Memon A. M., 2007. An empirical approach to testing web applications across diverse client platform configurations. *International Journal of Web Engineering and Technology (IJWET), Special Issue on Empirical Studies in Web Engineering*, **3:** 227–253.

[24] Elbaum S., Karre S., and Rothermel G., 2003. Improving web application testing with user session data. In *ICSE'03: Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, Washington, DC. ISBN 0-7695-1877-X.

[25] Gaur N., 2000. Assessing the security of your web applications. *Linux Journal*, **3**. ISSN 1075-3583.

[26] Halfond W. G. J., and Orso A., 2007. Improving test case generation for web applications using automated interface discovery. In *ESEC-FSE'07: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The foundations of Software Engineering*. ACM Press, New York, NY. ISBN 978-1-59593-811-4.

[27] Hao J., and Mendes E., 2006. Usage-based statistical testing of web applications. In *ICWE'06: Proceedings of the 6th International Conference on Web Engineering*. ACM Press, New York, NY. ISBN 1-59593-352-2.

[28] Hao D., Zhang L., Mei H., and Sun J., 2006. Towards interactive fault localization using test information. In *APSEC'06: Proceedings of the XIII Asia Pacific Software Engineering Conference*. IEEE Computer Society, Washington, DC. ISBN 0-7695-2685-3.

[29] Hassan A. E., and Holt R. C., 2002. Architecture recovery of web applications. In *ICSE'02: Proceedings of the 24th International Conference on Software Engineering*. ACM Press, New York, NY.

[30] Huang Y.-W., Huang S.-K., Lin T.-P., and Tsai C.-H., 2003. Web application security assessment by fault injection and behavior monitoring. In *WWW'03: Proceedings of the 12th International Conference on World Wide Web*. ACM Press, New York, NY. ISBN 1-58113-680-3.

[31] Joshi J. B. D., Aref W. G., Ghafoor A., and Spafford E. H., 2001. Security models for web-based applications. *Communications of the ACM*, **44:** 38–44. ISSN 0001-0782.

[32] Kallepalli C., and Tian J., 2001. Measuring and modeling usage and reliability for statistical web testing. *IEEE Transactions on Software Engineering*, **27:** 1023–1036. ISSN 0098-5589.

[33] Kasday L. R., 2000. A tool to evaluate universal web accessibility. In *CUU'00: Proceedings on the 2000 Conference on Universal Usability*. ACM Press, New York, NY.

[34] Kostoulas M. G., Matsa M., Mendelsohn N., Perkins E., Heifets A., and Mercaldi M., 2006. XML screamer: An integrated approach to high performance XML parsing, validation and deserialization. In *WWW'06: Proceedings of the 15th International Conference on World Wide Web*. ACM Press, New York, NY. ISBN 1-59593-323-9.

[35] Levi M. D., and Conrad F. G., 1997. Usability testing of World Wide Web sites. In *CHI'97: CHI'97 Extended Abstracts on Human Factors in Computing Systems*. ACM Press, New York, NY. ISBN 0-89791-926-2.

[36] Liu C.-H., Kung D. C., Hsia P., and Hsu C.-T., 2000. Structural testing of web applications. In *Proceedings of ISSRE 2000*, p. 84. ISSN 1071-9458.

[37] Martin E., Basu S., and Xie T., 2007. WebSob: A tool for robustness testing of web services. In *ICSE COMPANION'07: Companion to the Proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society, Washington, DC. ISBN 0-7695-2892-9.

[38] Murugesan S., and Deshpande Y., 2002. Meeting the challenges of web application development: The web engineering approach. In *ICSE'02: Proceedings of the 24th International Conference on Software Engineering*. ACM Press, New York, NY. ISBN 1-58113-472-X.

[39] Offutt J., 2002. Quality attributes of web software applications. *IEEE Software*, **19:** 25–32.

[40] Phillips B., 1998. Designers: The browser war casualties. *Computer*, **31:** 14–16ISSN 0018-9162.

[41] Ricca F., 2004. Analysis, testing and re-structuring of web applications. In *ICSM'04: Proceedings of the 20th IEEE International Conference on Software Maintenance*. IEEE Computer Society, Washington, DC. ISBN 0-7695-2213-0.

[42] Ricca F., and Tonella P., 2001. Analysis and testing of web applications. In *ICSE'01: Proceedings of the 23rd International Conference on Software Engineering*. IEEE Computer Society, Washington, DC.

[43] Ricca F., and Tonella P., 2001. Building a tool for the analysis and testing of web applications: Problems and solutions. In *Proceedings of TACAS*, pp. 373–388.

[44] Ricca F., and Tonella P., 2005. Web testing: A roadmap for the empirical research. In *WSE*. IEEE Computer Society, Washington, DC. ISBN 0-7695-2470-2.

[45] Ricca F., and Tonella P., 2006. Detecting anomaly and failure in web applications. *IEEE Multimedia*, **13:** 44–51ISSN 1070-986X.

[46] Rosmaita B. J., 2006. Accessibility first!: A new approach to web design. In *SIGCSE'06: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, New York, NY. ISBN 1-59593-259-3.

[47] Sanchez A., Vega B., Gonzalez A., and Jackson G., 2006. Automatic support for testing web-based enterprise applications. In *ACM-SE 44: Proceedings of the 44th Annual Southeast Regional Conference*. ACM Press, New York, NY. ISBN 1-59593-315-8.

[48] Sant J., Souter A., and Greenwald L., 2005. An exploration of statistical models for automated test case generation. *ACM SIGSOFT Software Engineering Notes*, **30:** 1–7ISSN 0163-5948.

[49] Scott D., and Sharp R., 2002. Abstracting application-level web security. In *WWW'02: Proceedings of the 11th International Conference on World Wide Web*. ACM Press, New York, NY. ISBN 1-58113-449-5.

[50] Sevilla J., Herrera G., Martínez B., and Alcantud F., 2007. Web accessibility for individuals with cognitive deficits: A comparative study between an existing commercial web and its cognitively accessible equivalent. *ACM Transactions on Computer–Human Interaction*, **14:** 12ISSN 1073-0516.

[51] Shams M., Krishnamurthy D., and Far B., 2006. A model-based approach for testing the performance of web applications. In *SOQUA'06: Proceedings of the 3rd International Workshop on Software Quality Assurance*. ACM Press, New York, NY. ISBN 1-59593-584-3.

[52] Sierkowski B., 2002. Achieving web accessibility. In *Proceedings of the ACM SIGUCS Conference on User Services*.

[53] Sneed H. M., 2004. Testing a web application. In *Proceedings of the 6th International Workshop on Web Site Evolution*. IEEE Computer Society, Washington, DC. ISBN 0-7695-2224-6.

[54] Stone R. G., and Dhiensa J., 2004. Proving the validity and accessibility of dynamic web-pages. In *W4A'04: Proceedings of the 2004 International Cross-Disciplinary Workshop on Web Accessibility (W4A)*. ACM Press, New York, NY. ISBN 1-58113-903-9.

[55] Velasco C. A., and Verelst T., 2001. Raising awareness among designers accessibility issues. *ACM SIGCAPH Computers and the Physically Handicapped*, 8–13ISSN 0163-5727.

[56] Wu Y., and Offutt J., 2002. In *Modeling and testing web-based applications*. George Mason UniversityTechnical Report ISE-TR-02-08.

[57] Xu L., and Xu B., 2004. A framework for web applications testing. In *International Conference on Cyberworlds*.

[58] Xu B., Xu L., Nie C., Chu W., and Chang C. H., 2003. Applying combinatorial method to test browser compatibility. In *Proceedings of International Symposium on Multimedia Software Engineering*, pp. 156–162.

[59] Xu L., Xu B., Nie C., Chen H., and Yang H., 2003. A browser compatibility testing method based on combinatorial testing. In *International Conference on Web Engineering*. Springer, Heidelberg.

[60] Zhu L., Gorton I., Liu Y., and Bui N. B., 2006. Model driven benchmark generation for web services. In *SOSE'06: Proceedings of the 2006 International Workshop on Service-Oriented Software Engineering*. ACM Press, New York, NY. ISBN 1-59593-398-0.