Modeling and Testing Web-based Applications

Ye Wu and Jeff Offutt
Information and Software Engineering Department
George Mason University
Fairfax, VA 22030, USA
(+1) 703-9934-1651 / 1654
{wuye,ofut}@ise.gmu.edu

Abstract

The Internet is quietly becoming the body of the business world, with web applications as the brains. This means that software faults in web applications have potentially disastrous consequences. Most work on web applications has been on making them more powerful, but relatively little has been done to ensure their quality. Important quality attributes for web applications include reliability, availability, interoperability and security. Web applications share some characteristics of client-server, distributed, and traditional programs, however there are a number of novel aspects of web applications. These include the fact that web applications are "dynamic", due to factors such as the frequent changes of the application requirement as well as dramatic changes of the web technologies, the fact that the roles of the clients and servers change dynamically, the heterogeneity of the hardware and software components, the extremely loose coupling and dynamic integration, and the ability of the user to directly affect the control of execution.

In this paper, we first analyze the distinct features of web-based applications, and then define a generic analysis model that characterizes the typical behaviors of web-based applications independently of different technologies. Based on this analysis, a family of testing techniques is discussed to ensure different level of quality control of web applications under various situations.

1 Introduction

Driven by the extreme demands of the business world and enthusiasm of the public, the internet has become indispensable to business, education, and even our personal lives. Thus malfunctions of the internet or in applications built on it can cause serious damages. For example, a glitch during an unscheduled maintenance at Amazon.com in 1998 put the site offline for several hours, with an estimated cost as high as $400,000. Even more costly, the relationship between customers and the company can be seriously damaged by such outages; the users do not care why the outage

happened, all they know is that the web site does not offer good service. Therefore, methodologies for adequately analyzing, constructing, understanding, testing and maintaining web applications will be essential, not only to the internet industry, but to American industry and commerce as a whole.

During the early days of the internet, it was primarily a typical client-server configuration with little flexibility or scalability that supported limited functionality and supported simple applications. In the last few years, this situation has dramatically changed. An "N-tier" model has been widely adopted to significantly improve quality factors such as scalability, flexibility, functionality, and availability. There are also a number of new technologies that are used in web applications. Nevertheless, nothing comes without cost. The techniques used to increase flexibility will also increase complexity, and the rapidly evolving technology provides new challenges to the techniques used to develop software. Questions of how to analyze, represent, test and maintain these types of applications have yet to be answered.

This paper first analyzes key differences between web-based applications and traditional software. One of the most difficult problems is the "dynamic" nature of web-based applications. This includes the rapidly changing technologies, frequent changes of user requirements, and dynamic aspects of the software technologies. These dynamic features can introduce several difficulties. For example, changes in technology may necessitate changes to analysis, testing and maintenance methodologies and tools. This paper proposes an analysis model that captures the dynamic features of web-based applications, and models the behavior in a technology independent way. This paper focuses on Java servlet-based web software, but the model is intended to apply to other technologies as well. The model first identifies the basic interaction unit between web servers and clients as *atomic sections*, which contain structural information with dynamic content. Then a complete interaction is modelled using compositions and transitions of these atomic sections.

The paper is organized as follows. The second section compares web applications with traditional software systems, then briefly reviews some of the existing modeling and testing techniques for web-based applications. Section 3 provides a technology independent model for web applications, and Section 4 discusses using the analysis model to test web-based applications. We offer conclusions and future research directions in Section 5.

2 Challenges and Related Work

One of the interesting challenges of web software is that web software has been found to have extremely high quality requirements [13]. Much of the software industry has been able to succeed with relatively low quality requirements; the combination of user expectations and market realities have been such that high quality usually has not increased profits. A combination of time-to-market and marketing strategies have almost always determined whether traditional software products succeed competitively. However, there appears to be little or no brand-name loyalty (that is, "site loyalty") for web applications. This means that if company A puts up a web site first, and B joins the market later, if users perceive B's site to be of higher quality then they will very quickly migrate to B's web site. That is, it is often better to be "later and better" rather than "sooner but worse".

This puts a great responsibility on the designers, developers, and testers to ensure that the web software exhibits very high **reliability**, **usability**, and **security**. These quality factors will

have a much stronger impact on company profits than for most "traditional" software. There are additional factors. For example, whereas a corner drug store ("brick and mortar") might expect to have customers form the neighborhood Monday through Saturday, 8:00 AM to 7:00 PM, a web-based company can expect customers from all over the world. It might be 3:00 in the morning in Virginia, but it's the middle of the afternoon in Beijing! Thus, web sites must have extremely high **availability**, not just 24/7, but 24/7/365. Another key difference is that, unlike shrink-wrap software applications, web-based applications do not have to be sold or distributed when updates are made. This, together with the rapid evolution of technology, means that **maintainability** is crucial for web software. Finally, unlike traditional businesses whose potential customers is typically limited by physical concerns such as geography and traffic, growth in web-based businesses has unlimited potential – there are currently hundreds of millions of users on the web, each of whom is only a click away and therefore "in the neighborhood" of the store. This means that web software must be highly **scalable** and ready to grow in terms of servers, services, and customers very quickly.

These high quality requirements bring new and interesting challenges to web software developers. This section first identifies these challenges, and then discusses some of the early research that has tried to develop ways to assure the quality of software that is used for web applications.

2.1 Challenges

Tremendous effort has been expended to assure the quality of traditional programs, resulting in testing techniques for both stand-alone and distributed systems. Although some of these techniques can be used to help assure the quality of web applications, some of the special features and requirements of web applications prevent them from being directly adopted. These challenges are summarized below.

1. The overall architecture of web applications are similar to client-server systems in many aspects, but there is a key difference. In traditional client-server systems, the respective roles of the clients and servers and their interactions are predefined and static. In web applications, however, client side programs and contents may be *generated dynamically*. For example, a server may return a dynamically generated HTML file that contains dynamically generated Javascripts, links and contents. This means that the subsequent interactions between the client and server depend on the previous inputs.

2. For traditional programs, *correctness* and *efficiency* are usually the most important quality factors. For web applications, other quality features can often be more important and yet we have few techniques for supporting them. For example, compatibility and interoperability are urgent and cause problems that are more serious than with traditional programs. Traditional programs are usually developed for a certain predefined, well understood environment, with very few conflicts and changes. Web applications often are affected by factors that may cause incompatibility and interoperability issues. For example, server components can be distributed to different operating systems, such as UNIX, Linux, Windows, MacOS, and AIX, each of which has multiple versions, and run with different web server packages, including IIS from Microsoft, Apache, WebLogic from IBM and others. The situation is even more complex on the client side, with different versions of web browsers running under

a variety of operating systems. Clients may also use different connection approaches, such as dial-up modems, direct internet access or wireless, and may also use different ISP providers. All of this *heterogeneity* makes it harder to produce web application components that are compatible with one another and that inter-operate easily and correctly.

3. Another difference between web applications and other types of programs is the variance in the control of execution of the application. For traditional programs, the control flow is fully managed by the program, so the user cannot affect it. When executing web application, users can break the normal control flow without alerting the program controller. For example, users can press the back or refresh button in the web browser, which totally changes the execution context, causing unexpected results. Furthermore, changes in the client-side configuration may affect the behavior of web applications. For example, users can turn off cookies, which can cause subsequent operations to malfunction.

4. Web applications also have much faster maintenance requirements than most traditional software. Web technologies evolve more rapidly than traditional software technologies, and the changes in web application requirements are also more dramatic. Therefore maintenance not only needs to be done more frequently, but needs to be done more efficiently due to the peculiar *time-to-market* pressure for web applications.

5. Web applications also have features that are not present in client-server and distributed systems. These include session control, cookies, the stateless aspect of HTTP, and new security issues. Therefore, new solutions are necessary for these special features.

2.2 Background

Although there are a number of differences between web software and traditional software, some testing approaches for traditional software can be adapted or modified [22]. Web applications tend to be based on gathering, processing, and transmitting data among heterogeneous hardware and software components so data flow approaches can be expected to be useful [4, 5, 6]. Most web software is object-oriented in nature, so inter-class [2, 3, 7, 9, 14, 17] and intra-class [1] This section reviews current research in these areas, and attempts to analyze the potential of traditional approaches to test web software.

Up to now, most web testing has focused on client-side validation and static server-side validation such as Javascript validation tools, link checking tools, HTML validators, and capture/playback tools. Research into functional testing of web software is just beginning. Kung et al. [10, 12] have developed a model to represent web sites as a graph, and provide preliminary definitions for developing tests based on the graph in terms of web page traversals. They define *intra-object* testing, where test paths are selected for the variables that have def-use chains within the object, *inter-object* testing, where test paths are selected for variables that have def-use chains across objects, and *inter-client* testing, where tests are derived from a reachability graph that is related to the data interactions among clients.

Yang et al. [21] focused on the called "three-tier" model, and developed a tool that helps to manage a test process across the three tiers. Their tool supports six subsystems to help track documents, monitor the processes, monitor tests, monitor failures and support test measurement. Their tool does not help generate tests or provide criteria for which tests should be designed.

Ricca and Tonella [16] proposed an analysis model and corresponding testing strategies. Their strategy is mainly based on **static** web page analysis and some preliminary dynamic analysis. As web technologies have developed, more and more web applications are being built on dynamic content, and therefore strategies are needed to model these dynamic behaviors.

Lee and Offutt [11] describe a mutation based approach to test XML-based web applications. Their approach focuses on the reliability of the data exchange among web component via XML, more specifically, functional correctness of the interactions. Nevertheless, the functionality of individual web applications is not considered in their work.

For the maintenance of web-based systems, most research has only considered static analysis of web page changes, such as adding or deleting web pages or hyperlinks [8, 15]. Other research has focused on maintenance of server-side components. With regards to rapidly evolving web techniques and increasing complexity of web-based applications, an integrated model for capturing the behaviors of web-based systems and supporting maintenance on these systems is necessary and urgent. To correctly maintain web-based applications, the relevant characteristics of web-based applications and the impact of these characteristics need first to be identified.

3 Modeling Web-based Applications

A key aspect, and perhaps the hardest part, of analyzing web applications is handling the *dynamic* nature of the software. The dynamic aspects are caused by uncertainty in the program behavior, changes in application requirements, rapidly evolving web technology itself, and other factors. The dynamic nature of web software not only brings challenges to analysis, testing, and maintenance, it also raises another important problem. When one part of the application changes, is it necessary to change every related component? This research attempts to address this question. We proceed by developing an analysis model that describes elements of dynamically created web pages using a regular expression notation. The analysis model can be used to support implementation, to develop tests, and to understand how changes to components will impact the rest of the application.

3.1 Analysis model

The major complexity of analyzing web software comes from the dynamic aspects, including dynamically generated client components, dynamic interaction among clients and servers, and the continual changes in the system context and web technologies. To accurately describe this dynamic nature, we start from the key element of web applications, the **interactions**. The model is then extended to induce additional features.

Even though interactions among clients and servers are indeterminate, there is a common theme that can be used. Web applications usually work in the following way: a client first retrieves information from servers in the form of HTML files, then sends requests to servers, and finally expects replies as HTML files. Of course not all interactions are done through HTML pages, but this mode of interaction currently accounts for the majority of web application processing. Thus this paper assumes that clients and servers interact through HTML pages. In the future, we hope to apply the concepts in this model to other forms of interactions such as XML-based applications.

To simplify the processing, we assume that each web application uses a portal, or a unique *start*

page S, which is an HTML document that initiates the navigation for the application. If there is more then one portal page $(S_1, S_2, ..., S_k)$, then we assume a unique start page can be created that has links to each actual portal page.

Beginning from the start page S, clients and servers interact with each other through requests and HTML files. In early web applications, most information was stored on servers as static HTML files, figures, and data files. As web technology has progressed, HTML files have been generated dynamically, by assembling pieces from separate files and program statements. To model the interactions of web applications, we first depict the basic elements that can be generated, then define a set of operations that can be used to generate new compound elements.

An *atomic section* is a static HTML file or a section of a server program that prints HTML. An atomic section has an "all-or-nothing property", that is, either the entire section is sent to clients or none of the section is sent. An atomic section may be a constant HTML section, or it may be an HTML section that has a *static structure* but may contain *content variables*. A content variable is a program variable that provides data to the HTML page but not structure.

For example, Table 1 shows a Java servlet from a server component P that is divided into four atomic sections.

p1 =	`PrintWriter out = response.getWriter();`
	`out.println("<HTML>")`
	`out.println("<HEAD><TITLE>" + title + "</TITLE></HEAD>)"`
	`out.println("<BODY>")`

p2 =	`for (int I=0; I<myVector.size(); I++)`
	`  if (myVector.elementAt(i).size > 10)`
	`  out.println("<P><B>" + myVector.elementAt(i) + "</B></P>");`
	`  else`

p3 =	`  out.println("<P>" + myVector.elementAt(i) + "</P>");`
p4 =	`out.println("</BODY></HTML>");` `out.close();`

Table 1. Servlet Atomic Section Example

The atomic section is defined as the elementary physical unit. When these units are combined together, more complex units are formed. The composition is usually done dynamically, and the actual composition is affected by the control flow of the server component. Possible compositions include *sequence*, *selection*, and *aggregation*. Formally, p is a *composite section* of a server program P if

1. *Basis*: p is an atomic section.
2. *Sequence*: $(p \rightarrow p_1 \cdot p_2)$: p_1 and p_2 are composite sections, and p is composed of p_1 followed by p_2.
3. **Selection** $(p \rightarrow p_1 \mid p_2)$: p_1 and p_2 are composite sections, and the server selects either p_1 or p_2, but not both.

4. **Aggregation** ($p_1\{p_2\}$): p_2 is included as part of p_1 when p_1 is transmitted to the client. For example, a function call in p_1 or a file inclusion command will include P_2 in P_1.

The simple operations of sequence and selection are extended with standard regular expression notation. p^* represents 0 or more composite sections concatenated together, and p^+ represents 1 or more composite sections concatenated together. p^n denotes exactly n composite sections, and $p^{(0|1)}$ indicates that p may optionally be included. Other expressions can be used as needed.

Atomic and composite sections define how HTML files can be dynamically generated. Given a server component, a *composition rule* is a regular expression that represents all possible complete HTML files that can be generated by the component. The composition rule for the above example is $P \rightarrow p_1 \cdot (p_2 \mid p_3)^* \cdot p_4$

The above representation can be used to model the internal structure of individual server components. To finish a complete transaction in a web application, the client and server components link the dynamically generated HTML files together. Interactions among client and server components are generally more complex than those of traditional applications in certain ways. Most traditional applications have deterministic function invocations; even the uncertainty caused by polymorphism is limited.

Web-based applications use HTML and action links to combine components. When HTML pages are generated dynamically, these links may rely on dynamic information, which means the contents are not known until execution time. Furthermore, users can modify the execution flow of web applications, taking some of the control away from server and client components. If a user hits the back button in the browser, the state of the application returns to a previous state, thus changing the control flow without notifying either the server or client components. To adequately depict these scenarios, our analysis model defines a *dynamic interaction*.

The interactions among different server components can be classified into two categories of *transition rules*, HTML links and actions by users. In the following, p and q are composite sections and s is a servlet or other software component.

1. *Link Transition* ($p \Rightarrow q$): Invoking a link in p causes a transition to q from the server to the client. If p can invoke one of several static or dynamic pages, $q_1, q_2, \ldots, q_k$, then the link transition is represented as $p \Rightarrow q_1 \mid q_2 \mid \ldots \mid q_k$. Link transition can be an HTML link defined via the <A> tag, or an action link defined in a <FORM> section.

2. *Composite Transition* ($s \rightarrow p$): The execution of s causes p to be produced and returned to the client. The servlet s will normally be able to produced several composite web pages, which can be represented as $s \rightarrow p_1 \mid p_2 \mid \ldots \mid p_k$.

3. *Operational Transition* ($p \rightharpoonup q$): The user can inject new transitions out of the software's control by pressing the back button or the refresh button. Operational transitions model these as well as transitions caused by system configurations, for example, the browser may load a web page from the cache instead of the server.

Based on the definitions above, a web application W is modeled as a triple $\{S, C, T\}$, where S is the start page, C is a set of composition rules for each server component, and T is a set of transition rules.

The following example is a small servlet program to provide online grade queries to students. A student must access the main page first to input an id and password. Then a servlet validates the id and password; if successful, the servlet will retrieve the grade information and send it back to the student. If unsuccessful, an error message will be sent back to the student asking the student to send an email to the instructor for further assistance. This small application includes an HTML file, a query servlet, and another servlet that processes the email to the instructor.

index.html:

```
<HTML>
<HEAD>
 <TITLE>Grade Query Page</TITLE>
</HEAD>
<BODY>
<FORM METHOD="GET" ACTION="GradeServlet">
 Please input your ID and password:
 <INPUT TYPE="TEXT" NAME="ID" SIZE="10">
 <INPUT TYPE="PASSWORD" NAME="PASSWD" SIZE="20">
 <INPUT TYPE="SUBMIT" NAME="SUBMIT" VALUE="SUBMIT">
 <INPUT TYPE="RESET" VALUE="RESET">
</FORM>
</BODY>
</HTML>
```

The corresponding `GradeServlet` is shown with its atomic sections in Table 2. `GradeServlet` uses three methods, `Validate()`, `CourseName()` and `CourseGrade()`, that are defined in `GradeServlet` but omitted for brevity. The start page, composition rules, and transition rules for `GradeServlet` are:

$S = \{\text{index.html}\}$
$C = \{\text{GradeServlet} = p_1 \cdot (p_2^* \mid p_3) \cdot p_4,$
$\quad \text{SendEmail} = ... \}$
$T = \{S \Rightarrow \text{GradeServlet},$
$\quad \text{GradeServlet}.p_3 \Rightarrow \text{SendEmail},$
$\quad \text{GradeServlet}.p_4 \Rightarrow S \}$

This definitions provide a preliminary model for web applications. The current model only includes Java servlet components. In the future we expect to extend the model to handle other technologies such as Java Server Pages, Active Server Pages, and XML.

If the link transitions are generated dynamically, then which composite sections are targeted can not be known statically. An analysis methodology will be developed to eliminate irrelevant composite sections, therefore deriving a more precise analysis model.

Applications of this model to issues such as testing, compatibility, and interoperability are introduced in subsequent sections.

3.2 Testing Web-based Applications

The model in the last section provides detailed information about the structure of web-based applications, including atomic sections, composite sections and their relationships. Before using

p1 =	`PrintWriter out = response.getWriter();`
	`out.println("<HTML>");`
	`out.println("<HEAD><TITLE>" + title + "</TITLE></HEAD>");`
	`out.println("<BODY>");`
	`if (Validate (ID, PASSWD))`
	`  out.println ("<B> Grade Report </B>");`

p2 =	`  for (int I=0; I < numberOfCourse; I++)`
	`  out.println ("<P><B>" + CourseName(I) + "</B>" + CourseGrade(I) + "</P>");`
	`  else`

p3 =	`  {`
	`  out.println ("Wrong ID or wrong password");`
	`  out.println ("<FORM METHOD=\"GET\" ACTION=\"SendEmail\">);`
	`  out.println ("<INPUT TYPE=\"TEXT\" NAME=\"SUBJECT\" SIZE=\"50\"");`
	`  out.println ("<INPUT TYPE=\"TEXTAREA\" NAME= \"BODY\" WIDTH=50");`
	`  out.println ("<INPUT TYPE=\"SUBMIT\" NAME=\"SUBMIT\" VALUE=\"SUBMIT\">");`
	`  out.println ("<INPUT TYPE=\"RESET\" VALUE=\"RESET\"></FORM>");`
	`  }`

p4 =	`out.println("<A HREF=\"index.html\">Return To Main Page</A>");`
	`out.println("</BODY></HTML>");`
	`out.close();`

Table 2. Atomic Sections of Servlet GradeServlet

the model to support testing, the execution of web applications must also be modeled.

In a web-based application, an execution of a test case includes a sequence of interactions between clients and servers; those interactions can be represented as a derivation. A *derivation* is a sequence of transitions that begins at the start page, and uses composition and transition rules to reach the desired page. A *subsequence* of a derivation is the sequence of transitions between two intermediate web pages.

A derivation for a normal grade query from the previous example is:

$S \Rightarrow \text{GradeServlet} \rightarrow p_1 \cdot p_2 \cdot p_4 \Rightarrow S$

There are several derivations for the case when a student enters an incorrect ID password pair.

$S \Rightarrow \text{GradeServlet} \rightarrow p_1 \cdot p_3 \cdot p_4 \Rightarrow S$
$S \Rightarrow \text{GradeServlet} \rightarrow p_1 \cdot p_3 \cdot p_4 \Rightarrow \text{SendMail} \ldots$
$S \Rightarrow \text{GradeServlet} \rightarrow p_1 \cdot p_3 \cdot p_4 \rightharpoonup \text{Previous S} \Rightarrow$
$\text{GradeServlet} \rightarrow p_1 \cdot p_2 \cdot p_4 \Rightarrow S$

Each derivation can be used to create a test case. For a complex application, the number of possible derivations can be very large and, if it is possible for the user to loop through a series of pages, infinite.

We assume web applications are based on the HTTP, which is a *stateless* protocol. This means that by default, a server's response is a function only of the current request from a client, so

previous requests will not be taken into consideration. Such stateless web applications simplify testing because relationships among multiple requests can be ignored. But stateless applications only provide limited functionalities, and are only useful for small, restricted application. Web applications can be made to be stateful by using technologies such as cookies. This allows the server's responses to be based on information from previous requests as well as state of the data on the server. This issue will be addressed in detail in subsequent work.

Another issue is that of output validation. When testing it is necessary to check the results of tests and determine if they are correct or not. Web software makes this harder by making parts of the outputs difficult to access. The simplest form of output validation is simply to check the result that is sent to the client. However, other parts of the output include state on the server, including files, data bases, and in-memory data stores such as session data and beans. Some of these outputs are hard to view and may be used by the same client in the same session, the same client in another session, or other users. This issue will also be addressed in subsequent research.

4 Conclusions

This paper has introduced a new technique for modeling static and dynamic aspects of web-based applications. The technique is based on identifying atomic elements of dynamically created web pages that have static structure and dynamic data contents. These elements are dynamically combined to create composite web pages using sequence, selection, aggregation, and regular expressions. A major advantage of this model is that it relies on the principles of the HTTP and HTML construction, thus is independent of software implementation technology.

An initial application of this modeling technique to testing web applications was also introduced. The testing relies on the descriptions of the web software behavior to define tests as sequences of user interactions.

We plan several significant research efforts using this model. An important step will be to develop algorithms that can automatically generate the atomic sections, composition rules, and transition rules from software. This is complicated by the heterogeneous nature of web software; we must analyze Java servlets, JSPs, and ASPs among others.

The model can be extended to represent data definitions and uses in the software. A number of issues in testing still remain, including the testing of *stateless* and *stateful* applications, as well as regression testing. We also plan to address ancillary problems such as automatically deriving test cases and validating output of web applications. This model will also support maintenance of web applications by providing a structure in which to describe and analyze software modifications.

References

[1] Roger T. Alexander and Jeff Offutt. Criteria for testing polymorphic relationships. In *Proceedings of the 11th International Symposium on Software Reliability Engineering*, pages 15–23, San Jose, CA, October 2000.

[2] M. H. Chen and M. Kao. Effect of class testing on the reliability of object-oriented programs. In *Proceedings of the Eighth International Symposium on Software Rliability Engineering*, May 1997.

[3] M. H. Chen and M. H. Kao. Testing object-oriented programs - An integrated approach. In *Proceedings of the Tenth International Symposium on Software Reliability Engineering*, November 1999.

[4] P. Frankl and E. J. Weyuker. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering*, 14(10):1483–1498, October 1988.

[5] M. J. Harrold and Gregg Rothermel. Performing data flow testing on classes. In *Symposium on Foundations of Software Engineering*, pages 154–163, New Orleans, LA, December 1994. ACM SIGSOFT.

[6] M. J. Harrold and M. L. Soffa. Selecting data flow integration testing. *IEEE Software*, 8(2):58–65, March 1991.

[7] H. W. Hong, Y. R. Kwon, and S. D. Cha. Testing of object-oriented programs based on finite state machines. In *Proceedings of the Asia-Pacific Software Engineering Conference*, pages 234–241, 1995.

[8] Chia-Lin Hsu, Hsien-Chou Liao, Jiun-Liang Chen, and Feng-Jian Wang. A web database application model for software maintenance. In *Proceedings of the Fourth International Symposium on Autonomous Decentralized Systems*, 1998.

[9] Zhenyi Jin and Jeff Offutt. Coupling-based criteria for integration testing. *Journal of Software Testing, Verification, and Reliability*, 8(3):133–154, September 1998.

[10] D. Kung, C. H. Liu, and P. Hsia. An object-oriented web test model for testing web applications. In *Proc. of IEEE 24th Annual International Computer Software and Application Conference(COMPSAC2000)*, Taipei, Taiwan, October 2000.

[11] Suet Chun Lee and Jeff Offutt. Generating test cases for XML-based web application. In *Proceedings of the 12th International Symposium on Software Reliability Engineering*, pages 200–209, Hong Kong, November 2001.

[12] C. H. Liu, D. Kung, P. Hsia, and C. T. Hsu. Structure testing of web applications. In *Proceedings of the 11th Annual International Symposium on Software Reliability Engineering*, pages 84–96, San Jose CA, October 2000.

[13] Jeff Offutt. Quality attributes of web software applications. *IEEE Software: Special Issue on Software Engineering of Interne t Software*, 19(2):25–32, March/April 2002.

[14] A. Parrish and S. H. Zweben. Analysis and refinement of software test data adequacy properties. *IEEE Transactions on Software Engineering*, 17(6):565–581, June 1991.

[15] F. Ricca and P. Tonella. Web sit analysis: Structure and evolution. In *Proceedings of the International Conference on Software Maintenance 2000*, pages 76–86, October 2000.

[16] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, pages 25–34, Toronto, CA, May 2001.

[17] C. D. Turner and D. J. Robson. The state-based testing of object-oriented programs. In *IEEE International Conference on Software Maintenance*, Montreal, Quebec, Canada, September 1993.

[18] Y. Wu, M. Chen, and M. Kao. Regression testing on object-oriented programs. In *Proceedings of the Tenth International Symposium on Software R liability Engineering*, pages 270–279, Boca Raton, FL, November 1999.

[19] Ye Wu, Dai Pan, and M. H. Chen. Techniques for testing component-based software. In *Proceedings of the 7th IEEE International Conference on Engineering of Complex Computer Systems*, pages 15–23, Skövde, Sweden, June 2001.

[20] Ye Wu, Dai Pan, and Mei-Hwa Chen. Techniques of maintaining evolving component-based software. In *IEEE International Conference on Software Maintenance 2000*, San Jose, Califonia, Octber 2000.

[21] Ji-Tzay Yang, Jiun-Long Huang, Feng-Jian Wang, and William C. Chu. An object-oriented architecture supporting web application testing. In *Proceedings of IEEE 23th Annual International Computer Software and Application Conference (COMPSAC 2000)*, Phoenix, Arizona, October 2000.

[22] Hong Zhu, Patrick A. V. Hall, and John H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, December 1997.