Strategies for Testing Web Applications from the Client Side

by

Lawrence Thaddeus Prevatte III

Bachelor of Science
Mathematical Sciences with the Computer Science Option
The University of North Carolina at Chapel Hill
1980

A thesis
submitted to the School of Engineering
at Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
May 2003

We the undersigned committee hereby recommends
that the attached document be accepted as fulfilling in
part the requirements for the degree of
Master of Science in Computer Science.

"Strategies for Testing Web Applications from the Client Side,"
a thesis by Lawrence Thaddeus Prevatte III

_____

J. A. Whittaker, Ph.D.
Professor, Computer Sciences

_____

M. Andrews, Ph.D.
Assistant Professor, Computer Sciences

_____

J. C. Wheeler, Ph.D.
Professor and Head
Electrical Engineering

_____

W. D. Shoaff, Ph.D.
Associate Professor and Head
Computer Sciences

Abstract

Strategies for Testing Web Applications from the Client Side

by

Lawrence Thaddeus Prevatte III

Research Director: James A. Whittaker, Ph.D.

This thesis is presented on the testing of computer software designed to operate over the World Wide Web. A fault model to guide Web software testing is developed, the components of Web software applications and their interfaces to each other are defined, and strategies to test Web software are discussed. The approach taken is to simplify the concepts of testing Web software to aid in a methodical treatment of the techniques required to conduct meaningful tests using a minimum amount of testing resources, conducting the tests from the client side of the Web application during the "ready for users" or acceptance phase. This text will present how the industry arrived at the current state of Web application quality, how the industry views Web applications, how to test them, strategies for testing Web applications from the client side, and interesting examples using the proposed methodology.

Table of Contents

List of Figures

Acknowledgments

I would like to express sincere appreciation to:

Dr. James A. Whittaker for the great part he played in the conception, research, and preparation of this manuscript and for his astute guidance in the thesis development process

Dr. Shirley A. Becker for providing constructive criticism of my literature search techniques and for facilitating discussions with graduate students developing automation tools for Web application testing

Madhan Thirukonda and Hyoung-Rae Kim for demonstrating to me Web application testing tools they developed and for discussing Web application testing with me

Dr. Carl I. Delaune for allowing me the opportunity to take his courses, for the knowledge he imparted to me on numerous occasions, for the encouragement he offered freely on many more, and for his advice to develop a thesis as part of my Master's degree

Dr. Arthur Dickinson for putting up with me in the majority of my coursework for this degree and for all the fascinating and invaluable information he has imparted to me as I have pursued this degree

The love of my life, my wonderful wife, Suzi, for enduring, encouraging, and enabling me as I've pursued this degree for the past three years, and for serving as a great example for going back to school after several years away from it

My son, Jonathan Frishman, for being supportive in my decision to go back to school while he was pursuing his own unique way through the world of academia

My parents, Rev. Thad and Martha Ann Prevatte, for the sacrifices they made to ensure that I was afforded every opportunity to acquire a quality education, for instilling in me the desire to do my best at anything I attempt, for helping me appreciate the value in obtaining a sound education, and for all of the other invaluable life lessons they taught me

My sister, Pam Land, for impressing upon me the importance of striving for, retaining, and improving upon the essential skills of common sense and social interaction throughout my life

My extended family, Craig Land, Nicole Land, John and Sue Fick, Brad and Wanda Fick, John Fick, Kate and Chris Walker, Amanda Fick, Casey Fick, Brian Fick, Kevin Fick, Betty Jo Prevatte, Mary Jo Kaczmarczyk, Andy Dresser, Pat Gilbert, John Kurowski, and Jeff Oliver for being so understanding and supportive of me as I undertook the pursuit of this degree

Dr. Fred Brooks, Jr., Dr. Stephen Pizer, Dr. Peter Calingaert, Dr. Don Parnas, Dr. Karl Freund, Dr. Dave Pargas, Dr. Donald Stanat, Dr. Carlo Ghezzi, Dr. Donald Richardson, Dr. William Shoaff, Dr. Joseph Wheeler, Carolyn Henninger, Sharon Stites, and all of the other professors and staff at the University of North Carolina at Chapel Hill and the Florida Institute of Technology who inspired me and helped me appreciate the fascination of the Computer Sciences

My coworkers and friends Joe Wehlen, Lyn Picou, Cathy Emerick, Lawrence Robinson, Mark Flom, Gary Meier, Frank Gutcher, Paul Halpin, Bob Henry, Pat Ruddell, Steve Hauss, Dr. Mike Sklar, Dr. Alan Drysdale, Jon Hillman, Dave Ungar, Colette Bessette, Nate Shedd, Dan Davis, Mike Shilkitus, Kevin Hoshstrasser, Georgiann Allen, Paul Seccuro, Bruce Birch, Doug Hammond, Pam Mullenix, Sue Waterman, and Paul Burke who encouraged me to pursue postgraduate education

My good friend Neil Scott for proving to me that it’s never to late to go back to school and for showing me the way

Chapter 1
A Fault Model to Guide Web Software Testing

1.1 The Purpose of Web Software Testing

There are several reasons to test Web application software. Web application software consists of programs and data that are designed to operate across the World Wide Web, a combination of network and client-server software and hardware that enable teleoperations between computers local to and remote from a user. Since a Web software application is first and foremost a software application, the same reasons for testing application software apply to Web software.

There are also considerations for Web applications that are different than other software applications or simply do not apply to other applications. First and foremost among these is the fact that a Web application is as much an extension of a company's or organization's "storefront" or office space as it is a system of software that performs some set of tasks. One of the most self-serving tasks a Web application, particularly a commercial one, performs is to keep the user at the Web site.

One could make the argument that encouraging the prolonged use of any application may not be in the best interest of the application because it merely increases the chance that incorrect or poor functionality will be discovered, but that's the price for success as well as the implied challenge for any software application. Any software application that performs poorly or malfunctions runs the risk of driving users to turn to alternative applications.

However, nowhere is this facet of application competition of more concern than in the world of the Web, where the user "is only one click away gone" [1][2] – either exiting from the Web altogether or navigating off of the current Web application and onto another, possibly competing company's Web site. [3] While this is of most concern to commercial Web application deployers, for any Web application: If the user loses interest in it, the functions that a Web application is to perform are superfluous.

While there should be no realistic expectation that even a perfectly functioning and performing Web application will hold the interest of all visitors to a Web site, you can make a darn good bet that a poorly and incorrectly functioning or performing Web application will drive away and discourage users faster than trying to catch flies with vinegar. Stress management consultants have identified a phenomena they've coined "Web rage", which manifests itself when users become frustrated with poor Web service and performance. [4]

Almost all of the users who were asked, in a survey concerning Web rage, what they were most likely to do when they didn't like a company's Web site indicated they'd move on to a competitor's site. None of the respondents planned to spend any more time on a bad Web site, although a few said they'd try to contact the company for help. The same Web users described frustrating Web sites as being hard to read (due to font choices), poorly organized, or out-of-date, or making it difficult to find a telephone number to use to get in touch with the company. [5]

The testing of Web applications in order to assure their deployers that users' experiences with the Web applications will be pleasant is therefore of extreme importance. Even internally deployed Web applications, such as Web-based training or corporate sites are useless when employees refuse to use them. So Web applications should be tested as often and as thoroughly as time, money, and other resources allow.

1.2 Types of Tests

Some of the more common types of tests are acceptance tests, conformance tests, usability tests, performance tests, reliability tests, and robustness tests. [1] Acceptance testing demonstrates the applicability of an application to the automation, usability, performance, reliability, and robustness requirements of an organization to the satisfaction of the organization. Conformance testing matches the implementation of an application to its stated requirements. Usability testing determines the ability of the application to meet the needs of users in performing their tasks in an automated fashion. Performance testing measures how much time and other system resources are required by an application to accomplish the tasks it performs. Reliability testing stresses the application in the face of system resource and other failures, including the ability of the application to continue to perform its tasks over a long period of time. Robustness testing investigates the application's ability to perform tasks in ways not originally intended and the application's potential and capacity to be modified and scaled beyond original requirements.

1.3 Testing Strategy

Whittaker defines various types of testing and describes a clear and concise approach to testing software, treating tests as attacks on the software under analysis and describing a battle plan of sorts for testing software applications. [6] A similar approach and presentation format will be taken in this text, however the primary focus of this document is on those characteristics and approaches unique to software applications designed to operate over the Web. Like Whittaker's text on "breaking" software applications, this paper will describe where and when Web applications should be tested but will focus on breaking Web applications from the client side. [6]

Adept testers who test Web applications at this point and have achieved mastery of discerning situations and scenarios that might cause Web applications to,

at best, perform inadequately may be referred to as “breakers”. A breaker might seem to be something out of a Web application’s developers’ worst nightmare, but I contend that a tester who can discover and report what any user might accidentally stumble across while using a software application will prove even more invaluable for a Web application’s developers than for developers of traditional software applications. This is because Web applications tend to be used more immediately and much more often than traditional software applications due to their being hosted on the Web. This text will also focus primarily on the areas of Web application software functionality and security, as they are most often viewed as the two primary foci of whether or not a Web application actually works correctly.

In contrast to a master’s thesis recently written by Jesper Rydén and Pär Svensson, titled “Web Application Testing”, this text will present examples and explanations of recommended attacks on Web applications. [7] The tests and examples follow the general style of presentation established by Dr. Whittaker for describing and illustrating tests on software applications. [6]

1.4 Information on Web Application Testing

Several books and other technical publications explore application software testing. However, very few comprehensive texts on testing Web applications existed by the summer of 2002, although several have been announced for publication by the fall of 2002. Most of the material found in the subject areas of testing Web application software is available on the Web itself, as are many of the technical and popular journals and other, traditionally print, publications.

Although a number of articles, papers, and texts treating various aspects of testing Web applications are beginning to appear (and a few have existed for several years), the amount of material available in this subject area pales beside the staggering volumes describing traditional software application testing. This situation

is anticipated when the subject area is as new as the testing of Web application software.

1.5 Thinking Outside of the Box

Those who have ever tested software applications are most likely familiar with the terms “black box” and “white box”, which refer to treatment of the application software as an opaque entity whose inner workings are hidden from the view of the tester (black box) and as a transparent entity whose inner workings (data structures, algorithms, etc.) are completely visible to the tester. Gray box testing is somewhere between the two. The gray box tester has some knowledge of the internals of an application, including algorithms, internal states, data arrangement, or program operation. [8] Just as there are many shades of gray, there are many diverse opinions on the amount of application-specific knowledge a tester should have when testing an application.

The type of testing this manuscript advocates could well be thought of as navy box testing, so named because a popular definition of “navy” as a color is “almost black”. Far from remaining completely oblivious to the type of application to be tested, a good tester should be capable of making some good guesses as to the types of algorithms, states, data organization, and program operation of an application.

A good tester should have gained through experience intuition as to the types of input and output data and algorithmic combinations that often prove troublesome for the type of application that is being tested. However, any deeper knowledge of the internals and implementation of an application, particularly a Web application, may actually prove detrimental for a tester. Tests designed to focus on particular implementation choices often have to be redesigned or rethought as subsequent versions of the application are released. Implementation-based tests may also suffer

from the tendency to test the commercially available development and runtime software upon which the custom-built portions of the application rely rather than on the custom-built portions of the application itself.

Web applications often employ client-side and server-side components, such as plug-ins, databases, and virtual machines, which were not developed by the Web application developers. While the correct use of these components and the way a Web application reacts when one or more of these components fail are important for a tester to probe, it is not the job of the tester to try to second-guess whether the Web application developer should have chosen one implementation over another. It is the task of the tester to determine whether or not the Web application designer and developer's integration of implementation choices on the client side and on the server side, i.e. the Web application itself, performs correctly and in a usable manner.

1.6 Understanding Web Software Behavior

In order to best determine how to test Web software, just as with any other software application, testers must understand the characteristics of Web software behavior in general and its interaction with other software and hardware. This means testers must study what a Web application does and the means it uses to operate properly. Arming themselves with this knowledge provides testers with loads of information on how to best spend their resources (time, money, and brainpower) to go after critical points in a Web application's armor.

Web software, like any other software application, has particular potential trouble spots, which will result in application or system failure if the application fails to properly anticipate and handle trouble. To a software application, trouble manifests itself in many ways, including bad input data, unacceptable combinations of good input data, invalid sequencing of events, and failures in system resource availability and response. Applications cannot avoid these types of problems. How

well a software application detects, protects itself from, and recovers from such occurrences is what separates sane applications from pathologic ones.

For Web applications, clumsy Web page and site presentation and navigation, insufficient handling of client-side, network, and server-side interface errors and fumbling data exchanges compound the opportunities to fail that software applications of any sort already provide. Good testers have either gained the experience, possess the amazing intuition, or both, to focus their attacks on software applications to devise tests that will cover the areas of most concern in the operation and performance of a software application as it attempts to perform its tasks. [6] Good Web application testers must also become well versed in the myriad of interfaces available to the Web application developer these days, due to the proliferation of various Web browsers, client systems, server systems, network systems, and client- and server-based Web software components.

To guide them in their bug quest, a fault model of a Web application is proposed. A fault model for a Web application describes in general terms the environment the Web application operates in, the capabilities of the Web application, how it is expected to be used by those whose requirements it was written to meet, and how it is expected to coexist with other applications in its world. The next section describes the general capabilities of Web applications and how they interact with other parts of the systems in which they reside. The following chapters describe strategies and tactics testers may follow to effectively explore aspects of Web applications most likely to expose faulty design and development of a Web application's implementation of its requirements (bugs in the application).

1.7 A Web Application's Environment

What is a Web application? And what makes it different in nature from any other kind of software application? Identifying the features of a Web application that

are similar to other types of software applications and the features that distinguish a Web application from other classes of software applications is a crucial step in determining how and where to apply testing resources in order to make the best use of a tester's time and energy. The types of interfaces and interactions with other components in the Web application's universe, and the nature of those components themselves figure heavily into generating a model of the environment in which the Web application operates.

1.7.1 How We Got Here

In the early years of programming, once the first few computers had been designed and the idea that programming involved both instructions and data had been established, machine designers had many ideas as to how computers would be programmed. One group favored designing machines that would be programmed procedurally, with instruction execution beginning with one instruction and moving on to the next instruction in sequence and with data defined and used by programmers in the same, stepwise fashion. Another group decided that machines should be designed with data in mind first, then instructions developed that would act on this data, with the machine being designed as an executive for the interactions of instructions that manipulate the data. The former type of machine was known as the Fortran processor, and the latter type of machine was known as the symbolic processor. (And, as happens often in industry, one company even took as their name the "symbolic" part of "symbolic processor".)

The Fortran processor designers plowed ahead with their machine designs, and the majority of the programming that was taught and put into practice was procedural. "Structured programming" came about when many in the industry came to the conclusion that procedural programmers were writing code willy-nilly, with what seemed like total disregard toward readability of program flow and other qualities that make code maintenance (repair and change) a little easier to perform.

Terms like “spaghetti code” described programs whose control flow entered and exited sections of code with little regard for code cohesiveness.

Although heavy use of and improvement in structured programming techniques and tools greatly improved the maintenance situation, many developers in industry rediscovered the idea that code could be more tightly coupled with data. As development tools and processing power were just beginning to reach the point where object-oriented programming became achievable, many developers and tool designers leapt onto the O-O bandwagon at full tilt, apparently believing that the sooner O-O programmed code could be developed, the sooner the improvement in code reuse O-O programming promised could be achieved.

While procedural programming had taken many developers and industry gurus many years to improve the reuse of code and to improve the state of the art in program design, implementation, and maintenance, many O-O proponents preached overnight delivery from low code reuse and increased programmer productivity. In writing to Dr. Fred Brooks, Dr. David Parnas explained how these expectations created the type of problems one might expect from such expectations:

> *The answer is simple. It is because [O-O] has been tied to a variety of complex languages. Instead of teaching people that O-O is a type of design, and giving them design principles, people have taught that O-O is the use of a particular tool. We can write good or bad programs with any tool. Unless we teach people how to design, the languages matter very little. The result is that people do bad designs with these languages and get very little value from them. If the value is small, it won’t catch on.* [9]

But Web application software development also sprang up during this time of adjustment, on the heels of the fast-growing development of the Internet and the World Wide Web. The nature of the differences of Web application software, and PC-and-network-based software in general, from software that existed before the advent of the Internet and the resulting World Wide Web has been recognized as an

inevitable consequence of the proliferation of the personal computer and large-scale inexpensive networking. In recognition of the new testing approaches that would be necessitated by the new operational and delivery models Web application development brought to application software, James Bach observed:

> *We've come to the point where the textbooks on software quality and testing must be rewritten. Most of them still sing the praises of white-box testing and 100% statement and branch coverage, which may have been fine for the mainframe world in 1976, but are completely impractical in the fast-moving desktop world of 1996. Testing can easily become an infinite task, so testing methods must be tailored to match the technological, economic and human factors in each project…Modern software consists, to a large degree, of components written by someone else. That means programmers can create more functionality while understanding less about it than ever before. Bad software has never been so easy to create.* [10]

In addition to increased code re-use without an accompanying realignment of developer responsibility or testing methodologies, development tools for Web applications have almost become so commonplace as to being built-in to every PC in every home. Add the hype that proclaims that any user can create Web sites to the seeming simplicity of creating Web sites, and the situation becomes nearly untenable. While the number of Web sites on the Web probably numbers well into the millions, the percentage of poorly designed Web sites has increased, as well.

Creators of personal Web sites may not need to worry much about offending Web surfers, but businesses are piling onto the Web in increasing numbers, and appear to be becoming much more concerned and aware of the dearth of quality Web site development that has been taking place. More and more, these days, businesses that deploy Web sites are requiring that testers perform more stringent testing and understand that the best testers are those who best understand how Web sites can be brought to their knees. In the mad rush to bring Web sites online as soon as possible, testers must understand the makeup of Web applications as well as which areas of

Web applications are at greatest risk of failure, because testing resources and time tend to be in limited supply.

1.7.2 The Makeup of a Web Application

Several views on the makeup of a Web application have been expressed. According to Nguyen, a Web application consists of Web pages, custom software that runs on the client system, and custom software that runs on one or more server systems. Nguyen maintains that not only are the browser, Java interpreter, client operating system, networks, and complex third-party products, database servers, and Web servers not part of most Web applications, but the expenditure of major testing resources on finding bugs that are generated from the components and services a particular Web application uses detracts from the primary purpose of the tester: testing the unique parts of the Web application. [8]

Stottlemyer views a Web application as a Web site whose functionality is intended to meet the business requirements, that is, the set of functions requested by the people with a vested interested in the development of the Web application. [11] Splaine and Jaskiel characterize a Web application as a class of Web sites that is but one of the applications utilizing the part of the Internet that is the World Wide Web. [12] Mosley views a Web application in a similar fashion, but also as an application belonging to the class of client-server applications in general. [3]

These and other technical treatments on the nature of Web application operation differ slightly, but they do help us understand the environment of a Web application by the properties they have in common. In straightforward terms, a Web application begins with Web pages served up to a client system by a Web server. The Web server responds to requests from a Web browser that resides on the client system. The Web server interacts with Web application custom code, the client-side browser, other client-side third-party components and system services, server-side

third-party components and system services, the networks over which these various pieces of software exchange data, and the Web navigation techniques employed by the Web application to perform its tasks.

1.7.3 “Hey, Kid, What’s With the Life Preserver?” [13]

Whittaker describes software applications as “residing completely in the care of the host operating system”. [6] On the client side of a network interface, Web applications are similarly encased under the auspices of the client-side browser. The browser allows the Web application to navigate within Web pages, navigate over networks to other Web pages, utilize client-side components, and move data over networks between the client system and server-side components. Figure 1.1 depicts this view of a Web application’s operating environment, modeled after Whittaker’s “life-preserver” view of a software application’s operating environment. [6]

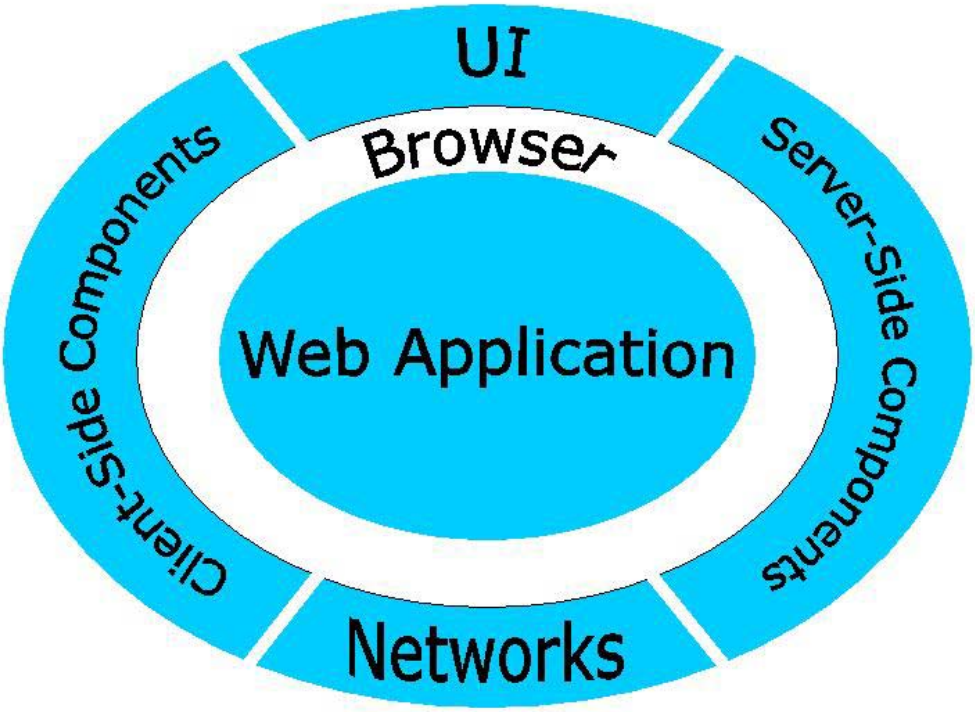

Figure 1.1 A Web Application's Environment

1.7.4 The Server Side

Following Whittaker's conceptualization of a software application's operating environment, each of the custom software components of the server side of a Web application is tested in four areas, each accessible to the application through its host operating system: the kernel, various APIs, the file system, and the User Interface (UI). Web application software performs the same basic operations that any other application software performs: reading data, processing data (performing calculations or translating input data to output data in other ways), writing data, and storing data in internal data structures. [6]

On the client machine, it makes very little sense to test most of these factors (you tend to end up testing how the Web browser reacts to system resource

deprivation rather than how the Web application itself reacts), but on the server side these factors not only affect how the custom, server-side components perform but also the performance and reaction of the Web application's client-side software. So although the client side of a Web application calls for a new model, testing the Web application for its reaction to server-side operations can mostly be accomplished using the familiar software application model applied on the server machine.

1.7.5 The Client Side

The Web browser is more than just the client-side software application that displays the Web application's Web pages and allows users to interact with the Web application through the use of various input devices such as mice and keyboards. It is the entire world to the Web application. Human users are in reality interacting with hardware input devices that communicate with the device services layer of the client system, which interacts with the Web browser through one or more set(s) of APIs (application programmer interfaces), including virtual machine interfaces, with the Web browser commanding sole access to the Web application through its Web pages. The Web application directs the browser to perform Web navigational and service tasks through agreed-upon protocols that are semi-standard across different Web browsers. In fact, the degree of compatibility between different Web browsers is an important consideration in the design and implementation of a Web application, because it determines how likely the Web application will be to perform its tasks on a given client system.

Note that testing the reaction of a Web application by invoking failures and near failures of system resources on the client side should be done to some extent, but a tester has to be very careful to not turn client-side testing into trying to break the Web browser instead of the Web application. Remember that the Web browser is a software application in its own right that runs on the client machine. Presumably, the

Web browser has been tested by its manufacturer and at any rate is not under control of the developer of the Web application.

1.7.6 The Human User

The human user portion of a Web's operating environment includes navigation within the Web site and to other Web sites, the user interface portion of a Web application, and the compatibility of a Web application with a given Web browser. Browser compatibility testing for Web applications includes validating that the HTML presented by the Web application to the language is valid.

This is especially important because Web browsers are not consistent in the way they handle HTML, much less in the way they handle incorrectly written HTML. How the Web application allows the user to move around from object to object on the Web page is important because users expect the user interface to be consistent from application to application, even though Web applications available to any user on the Web may have been written by developers all over the world.

Contrast this situation with non-Web software applications. Although non-Web software may be developed all over the world, non-Web software is far more likely to have been written in closer proximity to the user than Web software due to the differences in distribution methods (although this, too is changing – non-Web software is increasingly being offered via download over the Web, which although being different than pulling up the Web site of a Web application logically shares the distribution method of Web applications).

Web page navigation also includes provisions for providing data input values to and viewing data output values from the Web application via objects on the current Web page, whose importance will be discussed in section 4. Besides HTML forms and client-side scripts, input and output on Web pages may also be handled by client-side plug-ins and Java.

Surfing from Web page to Web page or window or even to other points on the current page may be provided for via active links on the current Web page, client-side and server-side image maps, plug-ins, scripts, and Java. Even if the method of navigation is not entirely consistent from browser to browser for the same Web page, navigation should be as consistent as possible both within the same Web application and between Web applications hosted under the same browser.

As an example, one popular navigation technique is the use of a drop-down list box and a "go" button. The user expects from the visual layout of these items that some item in the drop-down list is to be selected, then the user is to click the "go" button or press the "enter" key. Some browsers jump to the Web location corresponding to the selected list item only after the user clicks the "go" button with a mouse or presses the "enter" key, but other browsers jump to the selected list item location upon selection of the item in the pull-down list.

Web applications should also be developed with the expectation that the user will not have the "latest and greatest" Web browser loaded on their system, nor the latest client-side plug-ins (including the latest Java applet plug-in), nor will they necessarily have enabled every Web interpretation option, such as "allow Java", "allow all cookies", and "allow JavaScript". And the user's loaded browser may not support frames. Web applications should both anticipate these situations and provide convenient and user-friendly means to handle them. A tester must consider and test how the Web application handles many different types of client-side software configurations.

One particularly annoying example of this problem is how Web applications handle ActiveX. While some Web applications, most notably Microsoft's home site, actually do detect that the user does not have ActiveX capabilities, the most arrogant of these (like Microsoft's home site and MSN) instruct the user to find another browser to use that supports ActiveX (What they really mean is: "You should be

using Microsoft's Internet Explorer".) rather than providing a convenient means for the user to download a plug-in that will handle ActiveX.

1.7.7 The Network

The Web application tester must be careful about testing network problems as part of testing the Web application because some network-based tests on the client side, such as strangling network resources, may not actually test the Web application's behavior but that of the browser (depending on whether the Web application is redirecting to a URL or involved in some sort of messaging traffic, like HTTP, with the server-side part of the Web application). On the client side, more reasonable network testing includes garbling the information contained in or invalidating the incoming and outgoing network packets to see what the Web application will do in the face of bad input or output. The key is to devise tests that expose the behavior of the Web application itself rather than that of the Web browser.

The savvy Web application tester tend to ask the Web application questions like: "Okay, now what are you going to do if I change some of the data you're expecting from your server?" or "Alright, what if I drop, scramble, or slightly change your message to your server?" A good tester does not generally ask the browser: "What will you do if I bog down your view of the network so you time out trying to load the new Web page the user just tried to go to via an active link on the current Web page?" unless the tester is testing the Web browser itself.

1.8 A Few Words on Load Testing

Load testing, which essentially determines whether or not the components of a Web site (particularly the hardware) can handle a specified processing load, is an area of testing that is often misunderstood and, more importantly, incorrectly done.

While load testing a Web site is crucial to ensuring the Web site's success, this type of testing tests very little of the functionality of the Web application itself. Load testing is more for finding out how much load the Web site (its machines, systems, and software) can bear and still allow the Web application to perform in a timely fashion as advertised than for testing whether or not the Web application functions as required. [14]

One common misperception about load testing is that the testing is about what loads will break the Web site, particularly loads that place the Web site's hardware and server-side components under stress. This type of testing is more along the lines of stress testing, which should be done in addition to load testing, but its purpose is to discover what the Web application does in reaction to having its resources become unavailable and broken due to duress. [14]

Another of the most common misperceptions regarding load testing is that the number of concurrent users equals the load on the Web site. As the number of users of a Web application increases, especially as the number of concurrent users increases, the load on the application's site may very well increase, but it has been found that the two are not as tightly coupled as one would assume. [15]

Load testing is also generally performed by the Web site creators or their testers, because load testing is usually not considered valid unless the Web site layout, in terms of machines, software, and network components, are exactly duplicated in the test environment. Breakers will usually test Web applications from the perspective of the Web user on the client side of the Web site, which does not have the ability, resources, or access to exactly duplicate the server side of the Web application. [9][15]

In fact, Anderson claims that one of the top mistakes made in load testing Web applications is in starting too late. He observes that while many development projects schedule load testing after an application is complete, load testing should

begin as soon as the processes that make up the application are running, even before the user interface has been developed. Although he advocates performing load testing just before putting a Web site into production and load testing during regression testing after changes have been made to a Web site, he explains that conducting load testing during implementation is also useful for choosing between alternatives when considering what hardware the Web site should contain. [14]

1.9 Automation of Web Application Testing

Some degree of automation is essential to the testing of Web applications, due to the tight time constraints of testing Web applications, the ability of automated tools to perform relatively mundane and repetitive tasks accurately and quickly, and the availability of commercially available and free tools. [16] The Web Test Tools Web site describes and contains links to many of the free, commercially available, and fee-based tools for automating many of the testing tasks for Web applications. http://www.softwareqatest.com/qatweb1.html is the URL for the Web Test Tools Web site.

Automation can be overused if the tester is not careful. The creation of test scripts and parameterization of some automation tools can become a development project of its own, which can consume the time resources a tester would otherwise have available for actual testing. [16] Another consideration regarding use of automated tools is learning how to use tools will require time, and some of the tools available for testing Web applications can take a considerable amount of time to learn to use. [17]

Once created to test a Web application, automated scripts can be re-used for regression testing. [16] Indeed, testing tools have been developed that some may use primarily for regression testing of Web applications, such as a tool developed by Hyoung-Rae Kim, a graduate student at Florida Institute of Technology. This type of

tool parses a Web application and builds a profile that includes the pages of the Web application, the number of, types of, and links to objects on each Web page of the application that can be compared to a parse of the Web application after changes have been made. Analyses and comparisons of the data can be used to guide testers to pages that appear to be more complicated and to particular areas of the Web application that appear to have been modified. [18]

1.10 Understanding the Vulnerabilities of a Web Application

What a Web application does is pretty much what any software application does: it takes input, performs some computation on some data, and produces output. Beyond these standard software functional areas, however, Web applications exhibit vulnerabilities in the particular areas of functionality, security, usability, and client-side compatibility.

1.10.1 Functionality

The functional areas of particular importance to testers of Web applications include navigation, operations (computation), controls and displays, data handling (input and output), and error handling. These components are similar to their counterparts in other software applications, but their implementation in the more distributed world of Web applications can lead to some interesting, and sometimes confounding, problems as inputs, computation, and outputs may travel back and forth among components across the World Wide Web.

1.10.2 Security

Web applications face greater potential problems in terms of providing security to user data and to corporate data because of their existence on the wide-

open World Wide Web. User data must be protected from other Web users as well as from unauthorized personnel on the Web applications server-side machines. Corporate data on the Web application's server-side machines must be protected from unauthorized access on those machines, as with other software applications, but it is a much more complex problem to protect this type of data from users attempting to access the Web site's data via interfaces on the Web which were designed to allow wide-spread network access than from applications on the same machines or from dedicated client machines.

1.10.3 Usability

One of the most difficult aspects of Web applications to quantify, usability covers many of the capabilities as well as content arrangement of a Web application. While navigation layout and visibility, consistency, usefulness, and ease of use of a Web application may first pop into someone's head when thinking about the usability of a Web application, other factors in human-machine interface design are also examined by experts in this field, including accessibility. For example, the ability of people who cannot see or the ability of people who cannot hear to use Web applications depends on the implementations chosen and standards of usability that were followed when the Web application was created.

1.10.4 Compatibility

How the Web application interacts with the client-side machine can also turn a user's experiences using the Web from enjoying a pleasant Web experience to having a bad day. A Web application must function correctly in the user's Web browser, it must not damage the user's settings in the Web browser nor in the user's machine, it must not harm the user's machine environment, it must not disable the user's shared libraries, plug-ins, nor other, shared software, and it must not allow unauthorized access to any information.

1.11 Narrowing the Research Focus

As I conducted research into how to test Web applications, the following thoughts emerged and influenced how I narrowed the focus of my research. The first two thoughts I tried to classify were what the components of Web applications are and what the areas of most concern for Web applications are.

A Web application's UI and its use of client-side components appear to be functionally unique among other software applications because they are peculiar to operation over the World Wide Web, and are testable from the client side of the Web application. How a Web application handles security between its UI and its server-side components (distinct from how the Web application handles security on the server side) also appeared to be testable from the client side of the Web application.

The network, which is the World Wide Web, appears to be the independent component of a Web application. Web applications require the World Wide Web for interaction with the user, but the Web application developer has no control over the Web network itself. What the Web applications can do, however, is anticipate network conditions and ensure that the problem conditions between a Web application's client side and server side are handled by the Web application. Testing how a Web application handles network problems, however, can also be conducted from the client side of the Web application.

Server-side components appear, essentially, to be software applications running on one or more computers that communicate with the client side of a Web application. Functional and capacity testing of server-side components appeared to be best conducted on the machines on which they run as software applications. Texts describing software application testing include Dr. Whittaker's book "How to Break Software", which describes how to inject faults into software applications running on a computer, using testing tools like Canned Heat and Holodeck. [6] In addition, several tools exist for testing operational capacities of servers and server applications,

including load testing. However, such an abundance of security issues peculiar to the server side of Web applications appear to exist that they, too, seem to deserve their own, more detailed discussion.

Based on these thoughts, I decided to narrow the focus of this research to testing unique functional, security, and capacity fault areas of a Web application's interactions between the client side and the server side, with an emphasis on the functional areas. It became clear that many of these types of tests could be conducted on the client side of a Web application.

1.12 Summary

Software applications need to be tested for many reasons and throughout their life cycles, so testing manifests itself in many forms. Although Web applications, like other software applications, should be tested as they are developed, especially at the component level but also during integration and under load, the "ready for users" phase of a Web application is where a large number of the bugs users uncover will emerge. It is at this point that software testers highly skilled in attacking software are most likely to discover the lurking problems in functionality, security, usability, and compatibility that may make or break the Web application. The following chapters will present tests that expose problems in Web applications in these areas.

Chapter 2

The User Interface

2.1 User Interface Testing

The user interface is the component of a Web application that is most visible to a user. The user interface sets the tone and mood of a Web application and reflects the "face" a user will associate with the site owner. For an E-Commerce web site, the user interface serves both as the storefront and floor salesperson for the business. For other sites, the user interface serves as the host, curator, moderator, greeter, and other names that define the role of a person whose job it is to interface with customers, clients, and other visitors.

The user interface includes both the look and the feel of a Web application. The look of a Web application includes the layout of screen objects, the colors and patterns of text and graphics (including background graphics), the appearance of help text, and the appearance of screen control objects. Consistency on each screen and across the Web application contribute mightily to the overall appearance of cohesiveness across the site. The feel of the Web application – how dynamic screen objects react to user interactions with them and how the screens and screen objects are presented and changed – also determine the ease and applicability of use of the Web site for the user.

The term coined for how easy to use and how relevant the components of a Web site's screens are for users is usability. Usability also encompasses other aspects of Web sites. Increasing use of Web sites by people with visual, aural, and motor impairments has brought with it an increase in the attention to the need to design Web sites to be more inclusive of people with varying physical abilities

(accessibility). Research into usability is well underway across the world, and has been addressed by academic research programs, such as the E^3 Lab at Florida Institute of Technology, run by Dr. Shirley A. Becker, as well as by organizations such as the W3C, in their Web Accessibility Initiative (**http://www.w3.org/WAI/**).

So, testing the user interface is generally testing the Web application's usability. There are tools available for help in testing particular aspects of a site's user interface and there are also tools available which include these and other aspects of usability when reporting test results. While the tools range from commercial products to downloadable programs in the public domain, several sites also offer online tools. Online tools can be used by supplying a URL on the tools' Web pages. Some of the online tool sites report results right on the Web page, some send E-Mail messages containing results, and others post the information on their site, sending an E-Mail message that includes a URL to the information.

Several Web sites provide links to Web testing tools, including commercial, fee-based, and free tools that are available online or for download. Sites like the Software Testing Resources site, at **http://www.aptest.com/resources.html#web-source** and the Web Test Tools Web site described in section 1.8, which is part of the Software QA and Testing Resource Center site, located at URL **http://softwareqatest.com/** provide annotated links to various tools that can be used to help automate the tester's task of breaking Web software.

When a tester devises and applies this type of test, it's often helpful to "play user" and try to emulate a novice user's behavior, as well as behavior of a user with intimate knowledge of the Web application. Often, as with any other software program, a Web application's developer may unconsciously ignore error checking due to the "a user would never do that" mindset.

An example of the result of this type of thinking on an implementation occurred to the author many years ago, in the cafeteria of an NCR Engineering and

Manufacturing facility in West Columbia, South Carolina. If a user opened or otherwise fiddled with the transparent access door that allows the user to extract an ice-filled cup of soda from a drink machine, the sequence of cup drop, syrup and carbonated water pour, and ice addition often became disrupted enough so that the machine dropped a cup after pouring the liquids or dropping the ice.

The assumption the drink machine designers made was that since it makes no sense to extract the cup of soda until the syrup, carbonated water, and ice had been added to the cup, no one would have any reason to open the access door until the drink was "built". However, users being users, several bored users actually fiddled with the access door while waiting for their drink to be constructed. That drink machine taught a lot of young engineers the danger of assumptions – especially of the "a user would never do that" variety.

2.2 Test Number 1: Validate the Code

Background for this test

Web applications communicate to their users primarily through Web pages, which consist of Hypertext Markup Language (HTML) code. These pages may be hand-coded, generated completely by other software, or partially generated with hand-coded customization. Different versions of the HTML standard include different markup language codes. Web browser versions claim to correctly parse and display Web page elements and properties according to particular versions of HTML. There are other Web page description file types like style sheets (CSS) and extensible markup language (XML), scripting languages like JavaScript, and other Web page programming codes that Web browsers support in varying degrees, but HTML is the basic language that Web browsers read.

When HTML or other codes are incorrectly written, the Web browser that interprets a Web application's Web page may not correctly render the contents of the

page to the user as the page designer intended. Validating the code on a Web page means checking the Web page programming to ensure that it is correct according to a particular version of HTML (usually specified by the Web page designer within the Web page source code). Web page validation is one way to be reasonably assured that a Web browser that claims to correctly render a particular version of the code the Web page was written in will display the Web page content correctly to the user and handle the user's interactions such as mouse clicks in the way the Web page designer intended.

Rationale for this test

Web application developers, like other software developers, are only human, and can make human errors in miswriting codes. Even automatic code generators can miscode Web pages, either due to bugs in their programming or misuse by developers, although incorrect encoding is generally due to simply making a manual coding mistake or the result of a buggy code generator.

Results of this test

The most obvious detection of a Web page whose encoding is not valid is a visual one – some content of the page is incorrectly displayed or not displayed at all, some user interaction does not work correctly, or some action of the Web page such as navigation does not go as planned. Although any of these symptoms may result from other problems, invalid encoding is often the culprit.

Some errors in page encoding may not be so easily detectable, however, and may only be found upon examination or parsing of the page source code. Visual inspection can be made of the page source code, but automated tools for checking the source code are readily available and tend to be more reliable than human visual inspection, since visual inspection is prone to some of the same errors as human encoding.

Running this test

An example of invalid Web page encoding is an order status page on the Dell Computer Web site. Using a Web page code validation online service, such as the Web Design Group's (WDG) HTML validation service, available online at the Web address **http://www.htmlhelp.com/tools/validator/**, the tester enters the URL for the Web page whose HTML encoding will be validated, then the tester mouse clicks the "validate" button.

The result for the Dell computer order status page at the Web address **http://www.dell.com/us/en/dhs/topics/segtopic_orderstatus.htm** displays the URL of the page whose HTML encoding was checked, the date and time the Web page was last modified, the character encoding used by the Web page, and the HTML version the Web page claims to have been encoded to (HTML 3.2), and the errors and warnings found by the WDG HTML validator. Portions of the report are shown in Figures 2.1 and 2.2.

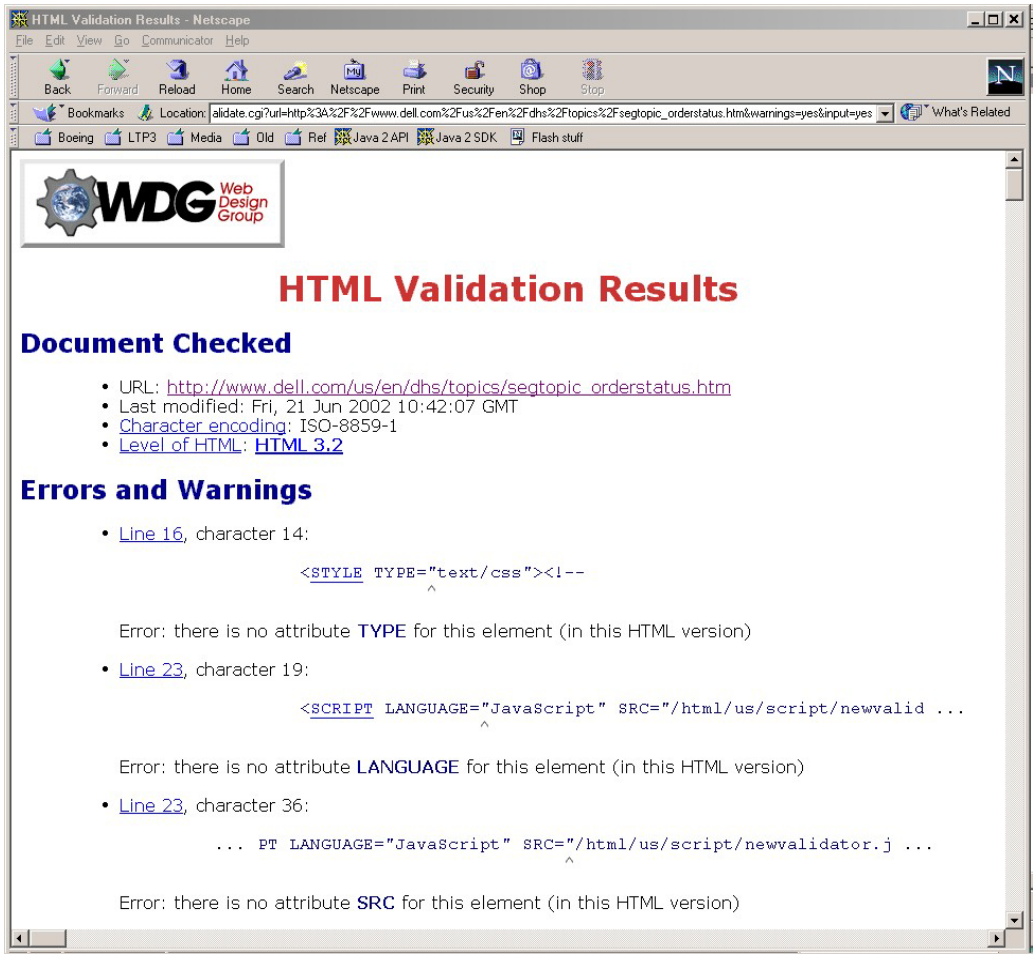

Figure 2.1 WDG HTML Validation Report Part 1

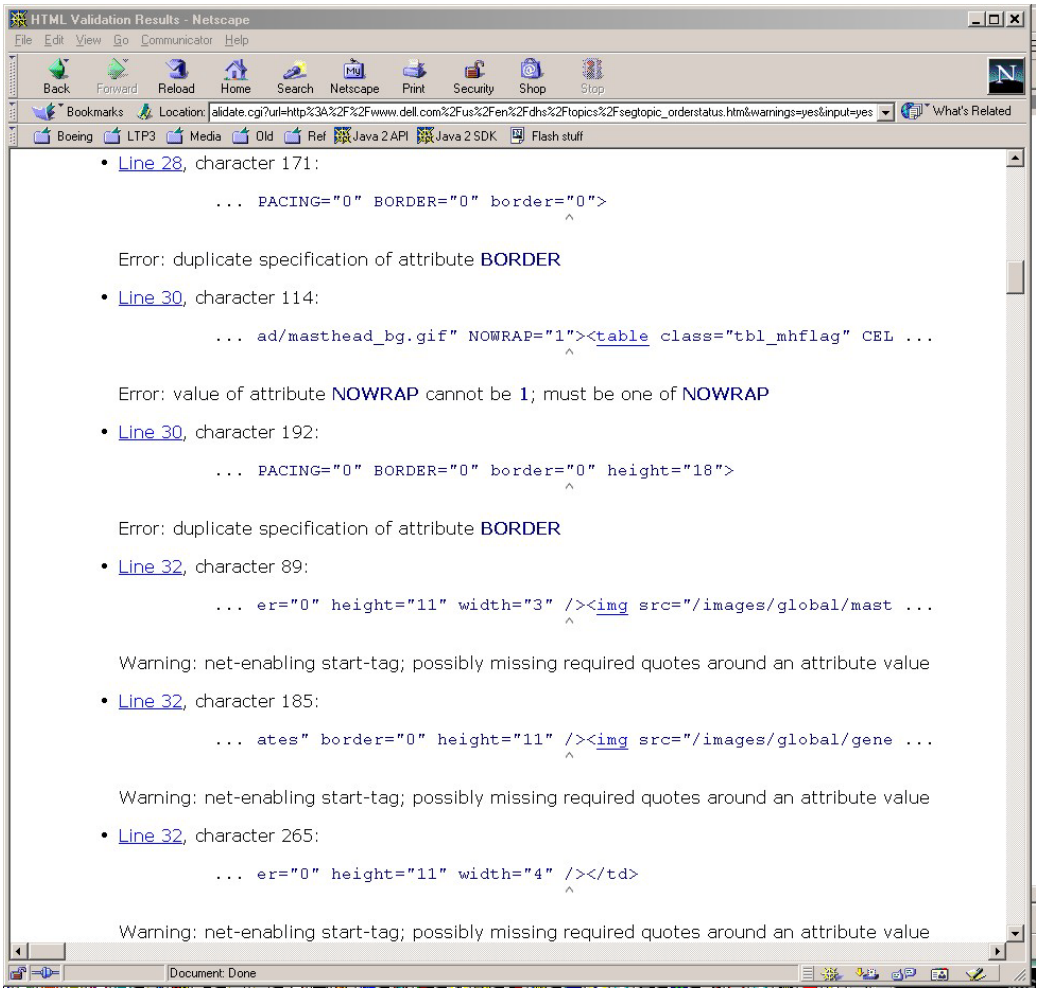

Figure 2.2 WDG HTML Validation Report Part 2

The WDG HTML validator reports that the Web page contains errors such as HTML tags that are not valid for this version of HTML (the “SPAN” tag, for example), attributes that are not valid for this version of HTML (the “TYPE” attribute on the “STYLE” tag on line 16), attribute values that are not valid for this version of HTML (the “NOWRAP” attribute on the “TD” tag on line 30 cannot be “1”), and duplicate attribute definitions for some tags (the “BORDER” attribute for the “TD” tag on line 28). The report also indicates that some tags contain an extra slash (“/”) just prior to the closing character (“>”) (an “IMAGE” tag on line 32). A

Web application tester that is familiar with different versions of HTML might conjecture from these hints that the first problem with the way the page is coded is that the HTML version the page claims to follow is incorrect.

Making use of another Web page coding validator service that is fee-based, NetMechanic (which is located at http://www.netmechanic.com/), the tester would receive a similar report, because the NetMechanic validator (HTML Toolbox) reports on the validity of the code for various versions of HTML. There are various Web page code checking tools and services available on the Web -- free, commercial, and fee-based -- each filling in different parts of the puzzle.

2.3 Test Number 2: Check Active Links

Background for this test

Web Applications use active links as one of the user-initiated ways to navigate from one area of a Web page to another, from one area of a site to another, and even off the site, to other sites that might or might not be related to the current site. Active links are generally visible to the observer because the text or graphic that makes up the link is usually designated by a different color than surrounding areas, with text usually also underlined or the graphic outlined. When a user moves the mouse over the text or graphic that makes up the link, the actual URL for the link is usually displayed in the status area of the browser.

Web page location targets – URLs that include page tags – are used to move the user to a particular location on a Web page, especially if the page is long enough to probably require the user to scroll to access parts of the page. Although tags to the top and to the bottom of a page are most common, tags often allow the user to navigate directly to a page location that is somewhere between the top and bottom of the page. Even though the entire Web page is available for the user to view when the page is rendered in the user's browser, the page tags provide a means for direct

access to particular parts of the page. Even when accessed from another page, an active link using a page tag may present the user with the desired portion of a page as it is brought in the browser.

If there is no active link to each area of a Web site, then if those areas are unreachable by any other means, such as form returns, users will never view those parts of the site, nor will users be able to participate in any interaction with those areas of the site. When this situation exists, then besides the confusion users may encounter when using the site, one would expect that electronic business sites would probably produce less revenue than with correctly functioning active links.

Rationale for this test

Incorrectly coded active links are usually the result of changes to the Web site, although they may also occur as developer or development tool encoding errors. Sometimes the active link is encoded into a Web page before the location tag or other Web page is created, which means the link is technically broken until the tag or Web page it refers to is instantiated.

When a Web application is modified, links to pages and page location tags that disappear or change must be modified along with the creation of links to pages and page location tags that are created as part of the modifications. Failure to accurately reflect these changes will also result in broken active links.

Results of this test

If the target of an active link does not exist, then one of two outcomes will occur that can be visually verified. When links to pages are broken, the Web server that serves the Web site will usually return an error status to the browser which the browser can then display to the user or the Web server may return a Web page that displays the error and may even provide help for the user in locating the information that the user was trying to locate. When a broken active link contains a page location

tag, the browser will not move to the portion of the page that the developer intended to be displayed. If the user is already on the page, then the browser usually shows no movement or moves to the top of the page. If the user is on a different page, then the browser usually brings up the desired page, but displays the top of the page rather than the desired area.

Running this test

Active links can be checked manually, but most Web applications of significant size contain many more active links than most testers would like to check by hand. Note that it is also not necessary to actually display a Web page just to verify that the page or a page tag on it exists. Automated tools are able to parse a Web page for its active links, navigate to those links, then parse the Web pages they find for more active links, navigate to those links, and so on, providing reports on the links that don't correspond to existing pages or page locations and therefore would not work. Tools to check for broken active links include the NetMechanic HTML Toolbox, the JimTool.com LinkChecker, Dr. HTML Pro (available at the Dr. HTML Web site http://www2.imagiware.com/RxHTMLpro/), as well as many other commercial, fee-based, and free online and downloadable tools.

2.4 Test Number 3: Check Alternate Text

Background for this test

One feature of the user interface a Web application presents to the user through a Web browser is called alternate text. Alternate text is text that is usually associated with graphical objects and is usually displayed by a Web browser in place of graphical objects that are not displayed, either because the Web browser cannot or is not configured to display graphical objects, or because the download of the

graphics have not yet completed. The alternate text is specified using the ALT tag of the graphical objects.

Alternate text is also read and rendered by vocalization software and hardware that translates text into sound. Originally developed to assist those who are unable to view Web content, text-to-voice translators are often used to allow multitasking power users to 'view' Web sites while they are doing other things at the same time.

Rationale for this test

Although there are many variants on how to use the alt tag, recommendations from Web site coding experts generally agree that this attribute should provide information relating to the purpose of the image rather than a description of the image. [19] Often, the tag is simply missing altogether, which provides no clue to the user how the image fits in with the remainder of the page content or the arrangement of the topic of discourse expressed by the page and image. Like height and width tags, which are also often neglected, resulting in Web pages that browsers cannot possibly even try to accurately display until the image itself is completely downloaded, image alt tags provide information to the user, the browser, and, as mentioned earlier, to associated accessibility hardware and software like vocalization applications.

Images are often used as format control within Web pages. Such images are usually very small, even single pixel graphics. Display or vocal rendering of these images may or may not be appropriate, depending on how the formatting function they serve contributes to the material the page is intended to convey. Even when a Web page author uses an image to separate page content and at is not appropriate to provide a textual alternative to the graphic, however, an empty alt text description is preferred to a missing description.

Web authors should also be especially careful as to surrounding content when specifying these tags, as even rudimentary vocalized "walk-throughs" of the page content can reveal incongruous discourse when the alt text is not well thought out. For example, if an image of a red text list "bullet" is coded as "red bullet", and preceding page content is similar to "Abraham Lincoln was slain by a …", then accessibility aids might render the combination as "Abraham Lincoln was slain by a red bullet …".

Results of this test

Examination of a report from a Web page validation testing tool or human examination of the content of the rendered Web page when image downloading is turned off in a browser can help a tester determine whether the text in the alt tags is provided, appropriate, or extraneous. Testing the page using accessibility software is an excellent way to get a feel for how the alt tags contribute to the topic of discourse of the Web page. Web page accessibility testing tools are also available. These tools provide reports specifically targeting accessibility areas of concern.

Running this test

Vocalization walk-throughs of the Web page content can reveal some problems with poorly coded alt tags, provided the tester has sufficient experience to anticipate how accessibility software might render the page content. As with many of the other attacks, automated tools provide more comprehensive and potentially more reliable checks than human inspection of the page content and source code. Most tools that validate HTML code also provide information on the content of alt tag text. The reports usually alert the tester to images that are missing the alt tag. Some tools list the alt tag text so that the tester can easily see the text and make a determination as to its appropriateness. It usually proves difficult to examine a Web page by for this test when image downloading is turned on, since most testers have a

fast enough Internet connection that the images on Web pages usually download too quickly to allow the tester to view and read the alt text for all of the images on each page. However, a tester can usually turn off image downloading for the browser that is being used, which will enable the tester to examine the text without the image downloading and replacing the text.

2.5 Test Number 4: Check Accessibility

Background for this test

Although the number of people who have some disability that hinders their access to the World Wide Web difficult (approximately 8% of the population), these people use the Web for the same reasons as anyone else: performing research, handling finances, shopping, learning, gaming, communicating, and conducting business. [20] There are obvious economic and goodwill advantages for all businesses and to businesses that cater to niche markets that may include those people who face technological access challenges. Providing access to as much of the population as possible also means that there are more people who can be informed, who can share opinions and knowledge, and who can be communicated with than just the non-challenged portion of the general population.

There are also legal reasons for considering accessibility that businesses and organizations need to consider when producing Web sites. Major companies and organizations that failed to have been successfully sued and have received public complaints. The National Federation for the Blind, for example, brought and won a lawsuit against AOL in 1999, claiming that AOL violated the Americans with Disabilities Act because its Web site was not accessible to the blind. [20]

Rationale for this test

Web pages encoded with formatting items that primarily present the content in a way that is visually pleasing does not always translate well to vocalization software. Complicated mouse and keyboard interactions, such as those necessitated by complicated image maps and pinpoint-accuracy-required mouse and link targets, are examples of problems for both blind-assisting systems and motor-assisting technologies. Motor-assisting technologies are used by people with severe motor skill challenges such as arthritis and the inability to quickly and precisely move fingers, hands, or arms. Improper and missing encoding of alt text for images, use of tables to control content formatting, and poor descriptions of link targets, titles, and headings, although providing sighted users with clarity, can very easily trip up blind users' access-assisting technologies, thereby reducing their access to the Web site. [20]

Results of this test

Visual inspection of link target, title, and heading text, visual and interactive examination of image maps and interactive precision required by the user, and visual review of alt text can be used as part of the testing for accessibility. Employing assisting technologies as part of the testing for accessibility can point out problems those with access challenges would face when presented with the Web site. Several automated Web site testing tools, such as WebSAT, which is part of the National Institute of Standards and Technology's (NIST) toolkit for testing Web site usability, contain tests for accessibility. Reports generated by this type of testing tool point out problems that assisting technologies would have rendering Web site pages. [21][22] Guidelines for accessibility are also available at the W3C Web site at URL **http://www.w3.org/WAI/**.

Running this test

Using an automated testing tool, such as WebSAT, provide the URL to the Web site or to individual Web pages, depending on the tool's ability to traverse active links. Examine the reports that result from running the tool to evaluate the page content. Also examine the Web pages themselves to detect potential problems with accessibility. Another tool for testing accessibility is Bobby, originally developed by at CAST (Center for Applied Special Technology), a not-for-profit group that specialized in promoting accessibility of computer technology. The CAST Web site is at URL **http://www.cast.org/**. Bobby is now owned by Watchfire Corporation, and is available at the Bobby Web site at URL **http://bobby.watchfire.com/**.

2.6 Test Number 5: Check Usability

Background for this test

Web site designers create sites for different users, and there are different criteria for usability of sites depending on who the intended audiences are. However, general categories and criteria of design consideration do exist, and most sites should be examined to determine how they measure up in these areas. Trade-offs such as load time versus content apply to sites with heavy graphics and graphic plug-ins, for example. Web sites that demonstrate or teach use of graphics requiring hefty plug-ins and example file downloads may require extremely high load times, but then the users of these sites expect that they may have to wait to receive some of the tools they may need to display and demonstrate the items they want to observe.

Commercial and other sites that cater to larger audiences are generally more concerned with the load times of their pages, and even Web sites such as the heavy graphic user sites mentioned above should follow enough of the usability guidelines

that are intended to help the user use the sites more easily and assimilate the information the sites are intended to convey.

The usability of a Web site includes how easy navigation and interaction with controls within and among the site's pages are to understand and use, how easy the content is to follow and how the content is arranged on Web pages, accessibility, and compatibility with browsers and other client-side tools. [23] Although personal taste will invariably enter into the design of Web site navigation, content, and formatting, attention to common guidelines is helpful in designing Web sites that are useful for as broad an audience as possible.

Rationale for this test

Failure of the site designer to consider carefully enough how different users will view and use the site can lead to coding site content that is unwieldy for browser viewing, navigation that is confusing or difficult, and the types of problems with accessibility and compatibility that are described in other sections of this paper.

Results of this test

Appearance, content, and navigation problems such as consistency and confusion are usually obvious via visual inspection. Whether the site is compatible with different browsers and platforms will appear obvious when the tester tries to render the pages in different browsers on different platforms, but some HTML validation tools can also provide reports on compatibility without the tester having to actually bring up the site within browsers to determine the site's page encoding problems. Form and navigation problems will show themselves when the tester attempts to navigate between form controls using the tab key and by using other, standard navigation techniques. The expected ability of a user to understand form objects and controls may be determined by visual inspection and by exercising the forms, although the number of and complexity of Web page objects in general can be

measured and reported by automated tools. Just as text can be rated as more complex depending on word usage and sentence structure, forms can be measured for relative complexity.

Running this test

Use reports generated by automated tools such as HTML validators, visual inspection, and interaction with the Web site to determine how well the site follows the general rules of thumb for usability. If the intended audience of the Web site is determined to be more or less demanding than the standard audience, then some plan that takes this into account should also be used.

General appearance guidelines to check would include the consistency of page layout, use of color, content formatting, and font usage. Page content should be balanced, with user attention drawn more to the more important elements of the pages, contact and other literary information easily visible and accessible, the purpose of the sight clearly understandable, graphics included where needed and not cluttering up the page where they're not needed, and the amount of content of each page sufficiently small enough to fit in common browser windows without requiring much scrolling vertically and preferably without requiring any scrolling at all horizontally at medium resolution (800 X 600). The content of the site should be examined to determine if the content contains any spelling or grammatical errors, if the readability level of the content is appropriate for Web access, if the content accurately renders the purpose of the Web site, if the content is current, and if a search function is included for the site, if appropriate. Graphical considerations include the quality of the images themselves, legibility of the text and images, and combinations of colors. [23]

The tester should navigate through the Web site to determine whether or not navigation and site structure are consistent, easy to understand, with clearly labeled links, consistency in the location of navigation among different pages, availability of

alternate forms of navigation (text links in addition to graphical links), and inclusion of navigation backwards as well as forwards within the site. One rule of thumb used for navigation is that it takes at most three (3) mouse clicks to move from any portion of the site to any other. [23]

The basic elements of usability should also be checked, some with automated tools and some with visual inspection and interaction. These include the logical arrangement and descriptions of form object and Web page controls, accessibility, correctly functioning controls and links on the pages, and sufficient explanation and directions related to the need for any special plug-ins, browsers, browser settings, or other helper applications.

2.7 Test Number 6: Verify Forms

Background for this test

When Web pages need to interact with users, requiring users to enter information via editable text boxes or configure information via checkboxes or radio buttons, the Web page designers often use forms. Once the information or configuration has been entered, the user is required to submit the form by pressing (mouse clicking) on a submit button. Other user actions can also be taken using form buttons, such as reset or clear the information on the form or navigate to somewhere else, such as checking out or continue shopping when using a shopping cart on a Web site.

Web page forms can be programmed so that form data can be checked for content and even checked for relative form data validity all on the client, or browser side. Usually, these checks are made when the user clicks on one of the form's interactive buttons.

Rationale for this test

Programming a Web page form is generally not all that tricky, but as with any type of software, programming errors can find their way in. When forms are programmed by hand, programming errors like initiating the wrong action for a button or incorrectly checking form data fields for validity can creep in by simply calling the wrong function or misusing a variable. Even with forms that are generated automatically by Web server page languages, however, button actions and form data verification data errors can occur.

Results of this test

Although some form programming errors can manifest themselves in very subtle ways, forcing the tester to painstakingly work backwards from an observed error to the culprit that originally caused invalid data to work itself into the Web application's computations, many bugs are immediately visible to the tester. Navigation errors are pretty obvious to even the casual observer, and even many data verification errors such as not accepting patently valid data will appear as obvious problems to the tester.

Knowledge of what the form is intended to do, and what the data or configuration of data on the form is intended for (the form's data domain) is of paramount importance for the tester. Once the form's use and knowledge about the data the form uses is determined, the tester should have a good idea as to what data is acceptable, what data should be unacceptable, how the navigation of the form is supposed to work, and how to determine if any of these functions fails.

Running this test

An example appears on an order status page on the Dell Computers Web Site (See Figure 2.3.). Ignoring that there are multiple order status pages on the Web site (which is an issue of consistency and is best dealt with in a section on testing for that

quality), there appear to be two forms on this page: one that allows a user to enter an Email address and password, and another that allows a user to enter an order number and verification data. From the information on the page, it appears that users who placed orders online will possess the former set of information, and users who placed orders by telephone will have the later set of data available for checking the status of their orders.

In this example, the online order information was entered and the submit button on the "Ordered Online?" side of the page was clicked. As the screenshot shows, the action taken by the form when the submit button on the "online" side is clicked somehow incorrectly checks the contents of the order number and verification data fields.

Examination of the page source code reveals that the two forms were generated, not hand coded, and one likely contributor to the problem is that the script function to check each form has the same name. One possibility is that the form generator was not designed to generate two forms on the same Web page, yet was used to do just that. The effect of this programming error is that a very obvious form bug appeared when apparently valid data was entered, and that the error affected both content validity checking and navigation.

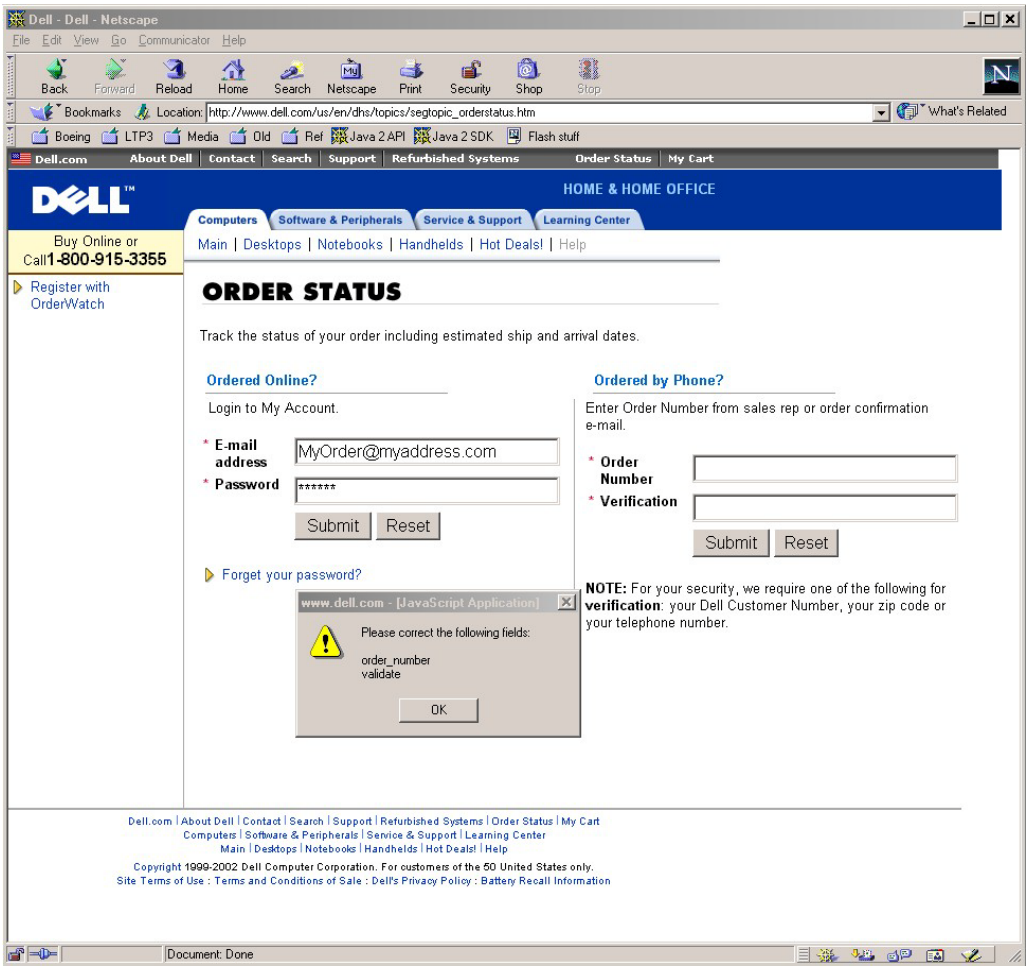

Figure 2.3 Dell Order Status Form Bug

2.8 Test Number 7: Enter Invalid Input

Background for this test

In manners resembling software applications that run on a machine, Web sites involve software that run on multiple machines. With a Web site, some form of computer software is accepting input data, whether it's a Web page, a form, a plug-in, a client-side helper application, a Java applet, a JavaScript script, or an application on the server. And just like a software application that runs on one machine, each

component of a Web site has the same potential for input data to create errors. The types of potential error conditions that regular software has can therefore be expected to be areas that should be checked on Web software: error messages, boundary conditions, repetition of the same data, features that share or interact with each other's data, initial condition problems, and value type problems. Any part of a Web site that uses data to operate, and particularly those Web site operations that use user-entered data, are potential places where errors can creep into the computational portions of the site. [6]

Rationale for this test

Even experienced developers may miss handling error conditions on input data, or may subconsciously make assumptions as to the order of operations without taking into account data interactions. Input data tests will uncover developer errors from overlooked cases, error messages that do not match the errors raised, failure of the developer to perform proper type checking, assumptions about data interaction or collaboration, inattention to default value combinations or provisions, and failure to either provide variables large enough to hold valid data lengths or to check for data of length that exceeds the anticipated input lengths and therefore variable sizes. [6]

Results of this test

Error messages displayed that do not appear to match the result of the improper data indicate that some particular state or condition case was not anticipated. This type of reaction can result from some part of the Web software not recognizing that the original problem with the input data occurred and may result from the software entering an incorrect state without proper data or at the wrong time. This type of reaction can be caused by any invalid data attack that is not recognized by the Web site, including type and length problems, boundary data problems, and data interaction problems. Repetitious inputting of the same, valid

data can result in error messages from site resource problems when the developer or the software's environment fail to release resources properly and resources used per instantiation of the data approach or overrun those available to the software. [6]

A lack of error indication when invalid data is entered can indicate that the input data was not recognized as invalid data but was somehow magically transformed without notification into acceptable data by the software, which is not an acceptable outcome. Invalid input data should be recognized and flagged by the software. [6]

Running this test

Attacks on Web site input data are carried out in a similar fashion to attacks on input data of other software applications. One difference to keep in mind between Web software and other software is that data entered on the client side may be partially validated at the client and then further validated on the server side, requiring more cooperation between software components than other software.

Since the tester is entering data and using Web page controls from the user interface, however, input data testing at the Web page level is still testing from the user interface. Try to enter data such as very low and very high values to test boundary conditions based on typical data types and sizes, such as the expected sizes of integer types, floating point types and the accuracy expected of various floating point types based on size. Enter numerical values that in collaboration with other data will produce potential problem computations such as divide by zero. Even enter data that is very near expected allowed minimums and maximums to see if computations using that data might produce errors. [6]

Try to enter enough erroneous values as to produce every error message that seems possible, based on the expected input and computations involving that data. For character strings, try to enter very long character strings, longer than one would expect to have as with first and last names, city and state names, and even freeform

text fields. Attempt to include characters that are not in the acceptable character set, if possible. Attempt to choose more than the allowed number of choices for fields that allow multiple choices. Attempt to leave fields blank, especially for data that one would generally always assume must be filled in. [6]

Repeat input of the same data in computational areas and in form entries. Try to update information using exactly the same data as was originally entered and accepted. Attempt to change already entered data that might be used as part of the lookup for other information. Enter data of the wrong type. For example, enter text characters in numerical fields, floating point values in integer fields, and integer values in floating point fields. [6]

Entry of input data can also be accomplished using automated test tools that record traffic between the Web pages and the Web server. These scripts can be edited so that input data values validated by Web pages can be changed to invalid data values actually being sent to the server. While entering input data using the Web pages tests the Web site's validation of input data at the page level, altering the network packets sent from the Web pages to the Web server tests the Web site's ability to handle invalid data on the server side. Most functional test tools include this capability, including WebART, from OCLC (Online Computer Library Center, Inc.)(http://www.oclc.org/webart/index.htm), WinRunner, from Mercury Interactive (http://www-svca.mercuryinteractive.com/), Solex, a plug-in for Eclipse, from Eclipse.org (http://solex.sourceforge.net/), HttpUnit, from SourceForge.net (http://httpunit.sourceforge.net/), QA Wizard, from Seapine Software (http://www.seapine.com/), MaxQ, from BitMechanic (http://www.bitmechanic.com/), e-Tester, from Empirix (http://www.empirix.com/), SilkTest, from Segue (http://www.segue.com/), Rational Robot, from Rational (http://www.rational.com/), and SiteTools Monitor, from softlight technologies, inc. (http://www.softlight.com/).

2.9 Test Number 8: Resize the Browser

Background for this test

Web pages can include content, including images, tables, lists, and paragraphs, formatting of the content, and background images. Viewing the pages in at one browser size, often influenced by screen size and resolution, even using different browsers, can lead to the false assumption that the Web site's pages will appear okay to users no matter how their machine and software are configured.

Rationale for this test

Web pages are often designed by software developers. Software developers usually develop software on fairly high-end machines, when compared with other computer users. Such machines tend to have large monitors, and developers tend to use very high screen resolutions on their monitors because this arrangement enables them to view more information on their screens' "real estate" than do lower resolutions. Some Web pages, although technically developed software, are designed by computer users who do not consider themselves software developers, and are not always "power users". Often, this type of developer maintains low screen resolutions and has a smaller monitor than a power user or a typical software developer. Even when Web pages are generated on the fly, the pages that the processes and tools that generate are not always checked for very small size, very large size, or for technologies such as WebTV.

A developer can fail to account for what happens when a user visits a Web site using a higher or lower screen size or resolution than the Web site developer, or uses a browser that is sized larger or smaller than the browser the developer was using when developing the Web pages. Unless the developer is either very careful or very lucky, rendering the Web pages of a site in a size other than the developer used to view the pages as they were being developed can produce visual results that were

completely unintended by the developer. The Web site could very easily be rearranged into a visual mess, or the content could be very hard to follow.

Results of this test

Here is one test that is easy enough to be confidently conducted by visual inspection alone. By resizing the browser and viewing the Web pages at different sizes, the tester can determine whether it is likely that some of the general users might have problems viewing the page content of the Web site. Pages that will require excessive scrolling by users with small browser sizes can be much of a problem as pages that display their content incorrectly because formatting or background graphics have rendered the content unreadable.

Running this test

The simplest way to test Web pages for size problems is to manually resize the browser window. In most systems, dragging one side or corner of the browser window with the mouse enables the user to expand or contract the browser. Test tools exist to help the tester automate this activity. But these test tools also help the tester know what actual size the browser is being changed to, in terms of pixels or resolution, something not always apparent from manually changing the size of the browser window. Two such tools are BrowserSizer, from ApplyThis Software, Inc., available at their Web site (**http://www.applythis.com/),** and Sizer, from the Brian Apps Web site (**http://www.brianapps.net/**). Using these tools, the tester can easily set particular browser sizes, including 640 by 480 pixels, 800 by 600 pixels, 1024 by 768 pixels. BrowserSizer even has a built-in WebTV setting.

2.10 Test Number 9: Examine Web Page Objects

Background for this test

The number and types of Web page objects, especially form items and controls, in part determine the expected amount of time a user will need to spend on a Web page to extract the information provided by the content on the page and to interact with the Web site, entering data and operating controls. How the Web site ranks in usability depends, in part, on the degree of user intuition and interaction required to navigate within and among the pages of the Web site, whether entered information is remembered and provided upon leaving a Web page and returning, the amount of help and hints provided by the pages, and the level of familiarity with the subject matter of the page's content. The arrangement of the objects on the Web site's pages should also be considered, as well as the frequency of page modifications due to the pages being automatically generated and due to updates to manually generated pages.

Rationale for this test

As Web site designers develop forms and other Web page controls, there can be a tendency to concentrate on the information the pages are intended to collect and how the information will be used by the site rather than paying enough attention to how the information is collected and describing how the page controls are to be used to enter or find the information. When this happens, the number of controls can grow beyond what is reasonable interaction for a user who is unfamiliar with how the Web site will use the information or as to how the different pages of the Web site interrelate. Also, when the Web site is updated, content creep can allow what were once fairly understandable pages to grow to pages with too many controls and form elements.

Results of this test

From reports generated by automated tools and from visual inspection of the Web site's pages, the tester can find links that are incorrectly coded, pages with too many form elements and other controls for most users to be expected to understand and use. The tester should especially be on the lookout for Web pages that require learning about, reasoning about, or remembering data not visible on the current page or form. While extremely large forms can often be broken up and presented across multiple pages at the site because different parts of a large form may really only be needed for particular subsets of data, forms that are concerned with related data that are split across multiple Web pages can also be confusing and difficult for the user to remember how to fill out.

Running this test

There are automated tools that can be used to check the number of Web page and form controls, their complexity, their functionality, and the amount of change they undergo when the site is updated. While useful for first time testing of Web sites, several tools function reasonably well as regression tools. Using data collected from the initial testing and measurement of a Web site, these tools compare the original site against further updates, producing reports on changes and additions in subsequent page rewrites. The tester should examine reports from automated tools and visually verify Web page objects and controls complexity and functionality.

One such tool, developed by graduate student Hyoung-Rae Kim at FIT, besides checking individual pages for objects and Web page complexity, compares new incarnations of a Web site with previous versions for number of errors and number of differences between the two versions. Links, content wordiness, and objects are checked, including reports on Web page and form controls such as buttons, select areas, text areas, and parameters. Besides the obligatory dead links

and empty anchors (code with no link encoded between the begin and end link tags), empty interface controls are reported.

2.11 Test Number 10: Check for Unauthorized Access

Background for this test

The Internet was developed as a means of sharing data. Assurance of privacy and security on the original Web was simple – there was none expected for anything that was put on the Web. However, over the years many companies, organizations, and individuals have sought to make use of the accessibility of their products and information over the Web. While there have always been individuals who have been concerned with the security of Web information, and privacy concerns have fast become foremost on user's minds as identity theft has become more common, Web security has lagged far behind the technological advances the Web has enjoyed over the past several years.

Security on the server side of a Web site includes the security measures put in place on the server side of the Web site by its system and network administrators, changing default passwords for applications and services the site uses, blocking access to machine ports that are not used by the Web site, and putting in place protective system and network firewalls and the like. There are also safeguards that Web site system administrators must take that prevent technically authorized accesses between applications on the site. Some of the major companies that are employing Web-based operations actually contract out security of their sites with Web security services. [24] Users of Web sites are likely more concerned about the privacy of their data and personal information than they are about any other security the operators of a Web site might have, so client data security should also remain at the top of the security concerns of the Web site developer.

Users have become paranoid enough about privacy and piracy on the Web that many of them have begun to install their own firewalls and intrusion detection software. Cookie data hijacking has become such a well-known source of hacked information that many users have turned on alerts every time a cookie is touched, and a number of users have blocked any type of cookie access whatsoever. If a Web site uses cookies or encoded equivalents injudiciously, or fails to adequately safeguard users' data from others, even on the client side, you can be sure that users will tend to shun that Web site.

Any site that performs E-commerce, requires user authorization or verification, or asks users for personal information must safeguard that information from other users and other sites. On the site's server side, users data should be protected from access by other applications or unauthorized operators.

Rationale for this test

Web pages that are participating in secure transactions should at the very least use encryption or other secure means of shielding the transactions from prying eyes, or in Web parlance, "sniffers' – tools that watch what is on the network and can decode open, unencrypted transmissions of data. Forms should verify types, sizes, and possibly ranges of values users enter, but verification against server data is more secure when that verification is done on the server than on the client.

When no encryption is used, the data and user information that is transferred between the client and server sides is out there for the world to see, and you can bet that someone's looking for just such unsecured data at any given time. Cookies that are created and contain unencrypted data and user information are available on the client side machine for any third party application or Web site to peruse at their leisure, particularly if the cookie is not destroyed immediately after it is set (which pretty much would defeat the purpose of most cookies, after all, since many are used to maintain state information between sessions or within a session but between Web

page accesses). While credit card numbers and other user data can be kept in cookies, it is preferable to transmit them in encrypted form to the server and let the server security protect them from other users.

Another potential security problem exists in how forms communicate data to the server. When the form uses the GET method, information from the form is sent to the server embedded in a URL, as opposed to the POST method, which hides the user information in an HTTP header. The POST method is generally considered more secure because it embeds the information within a header, but with either method, the file name of a Web page that receives the data can be sniffed by hackers. If the receiving Web page is openly accessible from the Web without verification, the data can be read by anyone. [25]

Results of this test

Determine whether Web pages supposedly involved in secure transactions are really locked in secure transactions, determine whether or not cookies contain nonencrypted personal information, determine whether or not the GET method of form data transmission is being used, determine whether or not transmitted form data includes Web page addresses, determine whether or not transmitted URLs contain unencrypted user information, and determine whether or not file names included in data transmission can be accessed by any user.

Running this test

When testing the Web site, verify that the sessions are run securely – that https is used instead of http, that the browser's security icon indicates that the browser is operating a securely linked Web page, and so on. Check cookies to see if any user data is contained in them in unencrypted format, including login names, passwords, and personal data like credit card numbers, addresses, and social security

numbers. Cookies are usually kept in a text file on the user's machine. This file can be opened in a text editor. If the data is text readable, it is not secure.

Using a network sniffer, examine the data packets (HTTP) between the client and the server. Look for unencrypted user information in the packets. Also look for URLs and file names on the server. Check (by entering the URL in the browser goto line) that the file names and other URLs are not accessible without verification from the client side. One good, inexpensive, and easy to use network sniffer is Analyzer, from Politecnico di Torina, available at their Web site (http://analyzer.polito.it/).

2.12 Test Number 11: Change Browser Settings

Background for this test

With increased user concerns regarding safeguarding user data, many users have become aware enough of the problems that cookies, JavaScript, Java, plug-ins, and installable downloads can cause that a large number of users have begun taking steps available to them to monitor and even prohibit interactions of these types with Web software. Turning off the use of these services, turning off image downloading, and even changing other browser preferences can make quite a difference in the way a Web site's pages appear or even work in a user's browser.

Rationale for this test

Web sites that do not take into account the user's ability to change browser settings may find that their Web pages no longer function or that they no longer operate the way the developer intended. If the user turns off cookies or changes the cookies setting to alert the user every time a cookie is touched, for instance, a page that makes heavy use of cookies may no longer operate or may inundate the user with so many cookie requests that the user abandons the page. Web sites that rely heavily on client-side scripting; that are designed to use client-side scripting in very

unconventional ways; make similar use of Java, plug-ins, or downloadable executable files; rely on font, color, images, or background graphics for content conveyance; and do not communicate well to the user how these items will be used run the risk of losing out to users who abandon the site for fear their data or other software may be compromised. What can also happen is that Web pages that appeared to make sense when rendered with the developer's browsers can turn into meaningless pages of seemingly poorly formatted content, depending on local browser settings.

Results of this test

The tester can check the Web site to see if information can still be transferred correctly between the server and client sides, if content still makes sense to the user when different settings are made in the browser, and if the Web site still performs correctly or at least explains to the user why it doesn't when certain browser settings have been made.

Running this test

Turn on and off, in differing combinations, browser settings including image downloading, JavaScript or Visual Basic scripting, and Java; block and turn on alerts to the use of cookies; and modify background, color, and font preferences, and observe the reaction of the Web site to these changes. Check if the Web site is still operating correctly, displaying content correctly, and offering help and explanations if certain settings required by the Web site for correct operation are turned off or set up differently than that required by the Web site to function and its pages to be rendered correctly.

Chapter 3

The Network Interface

3.1 Network Interface Testing

Testing the network interface primarily checks the server side of a Web site rather than the client side, although some attacks can be made from the client side. Network problems on the client side, such as heavy traffic on the Internet and other network situations that hinder traffic to the client, generally result in the user's Web browser not receiving responses within the browser's expected response window. The browser typically returns a timeout error message to the user when this occurs.

How the Web page on the client side handles invalid information from the server, however, can be tested, as can invalid information being sent from the client to the server. Because Web software should be designed so that the client side Web page does not generate invalid data to send to the server, and vice versa, injecting faulty data into the received and transmitted HTTP packets between the two sides can test how each side handles this type of situation, although this test is usually performed from the client side. And adjusting the speed of network traffic between the two sides, rather than simply causing packet loss due to timeouts, can point out potential timing related issues.

3.2 Test Number 12: Change the Network Access Speed

Background for this test

Users connect to the Internet at varying rates. High-speed connections are becoming more prevalent these days than in the past. Currently, cable connections and DSL connections serve those geographically lucky enough and affluent enough to have them, but many users still use dial-up connections. Although even dial-up speeds are higher than they were just a few years ago, individuals and businesses are still connecting to the Internet at very low speeds.

It's easy to forget that the LAN speeds that Web sites are often developed in are not how the population in general will access most sites. Often, the tendency in Web site testing is to investigate how the Web server will handle excessively high rates of demand rather than individually low speed access. Yet the degree of satisfaction of low speed access by a lot of people visiting a business's Web site may determine how well or how poorly the business may operate.

Rationale for this test

While the type of connection a user has to the Internet will, of course, affect how long Web site content will take to arrive at the user's machine and, minimally, how long the user's requests will take to head back to the Web server, there are Web site design considerations that tend to amplify the effect slower connection speeds have on the performance of the Web site, independent of data transmission rates. Database transactions, for example, that are designed to include Internet data transfer time can very easily time out when it takes an inordinate amount of time to move the information across the network interface. In general, any operation on the server side that includes transfer time across the Internet should be designed to operate at high as well as very low data rates.

Downloading large amounts of data should take into account the possibility that the data transfer and related acknowledgements may take a very long time just because the rate of data communication is very slow into some users' machines. Users are more inclined to allow downloads of large amounts of data to run in the background than they are to wait for excessively large Web pages to show up in their browsers. There are several ways to compress large amounts of data to reduce the time it takes to download, however, and there are even methods and online services that will compress Web page content and images. But consideration of the size of Web page content in the design of the pages can often make the download of the pages more tolerable for the users with low connection speeds. No matter how amazing and wonderful the images, graphics, and sheer volume of content on a Web page may be, if it's not being perused because users have abandoned the page due to excessively slow page loading time, all the fanciness and information in the world won't make up for the loss of a significant number of potential customers.

Results of this test

Visually comparing the download time of the Web site's pages to that of other Web pages with similar content at different connection speeds enables the tester to get a feel for the relative wait time users with similar connection speeds would have to wait for the Web site's pages to be rendered. Reports generated by automated test tools, such as NetMechanic's HTML Toolbox, can provide the tester with absolute page content sizes and even approximate download times at different connection speeds.

Running this test

To visually compare the Web site's page download times relative to those of similar Web sites, call up each of the pages and manually time how long the browser takes to render them. Using this method, the tester must keep in mind that only

relative speeds mean much and that download times may be dependent on networks proximity and routing from the Web site to the user's machine.

Using an automated test tool, enter the URL(s) of the Web site pages to have the tool generate reports on the size of the Web page and the estimated download times dependent on different connection speeds. Use tools like Canned HEAT, from FIT's Center for Information Assurance, degrade the network access speed to simulate lower connection rates. Simulate user access at lower rates and test page loading, navigating among, and interacting with pages from the Web site to verify whether or not the Web site can handle activity at slow speeds.

Chapter 4

The Client-Side Components

4.1 Client-Side Component Tests

In addition to the browser, which renders user interface elements of a Web application, other software on the client side of the application are often utilized. These client-side components should not be overlooked when devising a test strategy for Web software. Failure of the Web application under test to correctly and safely utilize its chosen client-side helpers should be considered just as much a failure of the Web application as a breakdown in any of its custom code.

4.2 Test Number 13: Check Plug-Ins

Background for this test

Some Web pages require plug-ins, which are common helper applet components share among Web site designers, to display or otherwise render Web page content. Applets are available for rendering of audio and visual content, for execution of code on Web pages like Java applets and ActiveX objects, and for even more specialized but common operations like keeping track of software version updates.

Whenever a Web page's encoding requires a plug-in to operate on content, the plug-in generally checks to see if an instance of it is loaded in the browser displaying the page. The version of the currently loaded plug-in can also be checked to verify that it is sufficient to render the specified Web page content or if the plug-in needs to be downloaded and started. In the old days of the Web, Web pages usually

contained a message to the user that the plug-in needed to be downloaded and installed, requiring termination of the browser before the plug-in could be installed. Today, however, many plug-ins are capable of being automatically installed while the browser is running, which means that the user doesn't have to handle recording a link to the page that requires the plug-in, manually shutting down the browser, manually installing the plug-in, manually restarting the browser, and then surfing to the Web page that contains the content.

Rationale for this test

Not all Web sites and not all browsers in use handle installation nor recognition of plug-ins as seamlessly as just described. In addition, some plug-ins generally come pre-installed in some of the browsers these days, and since not all users have these browsers and since even pre-installed plug-ins can be uninstalled, intentionally and accidentally, a Web site should be tested to see whether there have been any unexpected assumptions about plug-in availability that will hinder the operation of the site. Some sites have been developed to use their own, proprietary plug-ins.

No page of a Web site should be designed with any assumptions as to which part of the site is initially visited by the user, as users can enter sites from any page that can be linked to from elsewhere. If a user enters the site on a Web page that uses the site's plug-in but does not handle checking and installation, assuming that these are handled from introductory pages of the Web site, then the Web page the user entered on will have a difficult time using the plug-in.

Results of this test

If plug-ins are not available at the client's browser that a Web page expects to use, the Web page should let the user know that the content requiring the plug-in will not be rendered. The Web page should also help the user obtain and install required

plug-ins, and can possibly provide another method for the user to obtain or view the content that requires the plug-in.

Running this test

One test to check for plug-in operation is to test the Web site with different, freshly installed Web browsers. What works in one browser and even with different versions of the same browser may not work with other browsers and versions.

Another good check for plug-ins is to uninstall all plug-ins from Web browsers before bringing up the Web site. Pages that are encoded to user plug-ins should recognize that the required plug-ins are unavailable and should provide help to the user for loading the plug-in or obtaining the content without it, if possible.

4.4 Test Number 14: Check the Environment

Background for this test

When a Web site's client-side Web pages begin to be rendered by the user's browser and the user begins interacting with the Web pages, resources begin to be consumed on the client. The browser can quickly consume resources available on the client, including file, memory, and process resources as it attempts to handle resource requests from the Web pages. While the browser controls the environment the Web pages of a site operate within on a client machine, user interaction with the Web application can thwart the protection to the client that the browser provides. When the user accepts downloads and installation of downloaded software from the Web pages, control of client resources shifts from the browser to the user.

Downloads, installations, and program executions initiated by a Web site's client pages and accepted by the user can affect the client machine's resource availability, even when there is no malicious intent on the part of the Web site. The user becomes responsible for ensuring that there is enough file space, memory, and

process resources to handle the additional load imposed by this activity. Of course, the client machine's configuration is the user's responsibility, but Web applications should consider the burden that could be placed on the client machine when they are recommending and instructing the client to download, install, and execute software originating from the Web site or from third parties.

The possibilities of configuration problems, licensing nightmares, and resource degradation on the client machine are some of the reasons many companies develop policies prohibiting the practice of downloading, installing, and executing software from the Web by employees. The problem is that when a Web site recommends or requires these types of operations in order to render the content of the site, users must decide between violating rules put in place for sound business reasons versus receiving and viewing the content offered by a Web site. Besides the problems encountered from malicious Web sites, when buggy software from otherwise trustworthy sites negatively affect the client machine, the creation of problems on the client machine rest heavily on these Web sites.

Rationale for this test

Client-side errors in the way data is handled by Web pages, including gobbling up memory and process resources, usually stem from a lack of sufficient testing to expose these types of problems as the Web site is developed. Insufficient testing of Web pages under different browsers will often fail to identify features of the way different browsers handle Web page data and resource requests on client machines. And while not every configuration possible on the client end can possibly be tested when developing a Web site, testing of downloaded software installation and execution, for example, can reveal potential sources of problems such as unchecked overwriting of client-side shared libraries and DLLs.

Results of this test

Overuse and general browser inability to handle resource requests are often evidenced by browser failures to operate properly. For instance, many of the popular browsers such as Netscape and Internet Explorer begin experiencing toolbar icon display problems when the Web pages they are rendering cause resource availability to drop below levels that allow the browsers to function properly. The same types of problems – toolbar display problems and operational problems – with other software running on the client machine at the same time of the Web session may also be visible to the casual observer or tester.

Problems with overwritten shared libraries and DLLs may not surface until long after the interaction with the Web site is over, as other applications on the client machine that depend on these common code modules are run and begin exhibiting strange behavior and crashes. One way to determine whether the interaction with a Web site causes such problems is to compare client machine configurations, including shared code module versions, between the state of the client machine before and after a Web site has been tested. Although there are tools available that can record machine configurations and the reports can be compared after any test of a Web page, any time there is interaction by a Web site that switches the responsibility of the machine's configuration from the browser to the user, the client machine configuration should be considered suspect.

Running this test

Test the Web site for environment problems by opening many multiple browser sessions of the Web site, by running within the Web site for extended lengths of time, and by checking the Web pages for potential resource-draining data usage and operation. Study suspicious browser and other application operation as potential resource draining situations. Take special note of any of the Web pages that request that the user download, install, or execute software outside of the control of

the Web browser, and compare the initial client machine configuration against the configuration of the client machine following such operations, paying particular attention to shared code modules additions and version changes. Examine any discrepancies as potential problems from interaction with the Web site.

Chapter 5

Server-Side Components

5.1 Server-Side Component Tests

Here are tests beyond the standard security, denial of service, and load and sizing tests that can be conducted on the server side of a Web application (load testing was discussed in Chapter 1). Server-side tests are conducted on the Web site's server and associated local network elements. Major commercial tools designed to test Web sites concentrate on this area of a Web site because the intervening network elements between the client and the server can vary from session to session on the World Wide Web and because so much of how the Web site's pages and client-side helper applications and plug-ins interact with the user depend on the user's machine and software configuration on that machine.

Performance and functional operation of applications and services on the server side, however, are of paramount importance to how the Web site works regardless of the configuration and operation on the client side, and no amount of improvement to the client end of a Web site is likely to overcome major problems with the server side of a site.

5.2 Test Number 15: Test Concurrency

Background for this test

Concurrency, with regard to Web site operation by users, has at least two aspects that must be examined. Of obvious importance to the Web site tester is how many users the Web site can service at once. While the number of different users

that can use the Web site at once is important, also of interest is whether or not the same user can maintain multiple sessions on the Web site at once, on the same machine as well as on multiple clients. Besides users who may be logged in to multiple machines at the same time, user IDs may also be share among several different persons who may be logged in to the Web site concurrently.

Often regarded as an input factor for performance testing such as load testing, the number of concurrent users is considered by some performance testing experts as a result of load testing. As load testing is measured over time, some even suggest that a better input for load testing than number of concurrent users at a given time, which denotes no real time measure, should be the number of user sessions started per unit of time. [14]

Testing for concurrency itself, then, involves finding out how many concurrent user sessions the Web site can successfully service. If the site cannot accept any more user sessions after some are in progress, or if the site cannot maintain Web page load times within user-tolerable limits, or if the site cannot allow a user to log in more than once at one time, then the Web site's ability to handle concurrent users should be brought into question.

Rationale for this test

Since Web sites are often made up of many components, including the Web server and software that is serviced by the server and Web pages of the site, the component or components with the lowest number of allow concurrencies will often be the bottleneck(s) of the site with regard to concurrency. The correlation is not always as clean as finding the particular service with the least concurrency, however. What determines where the bottleneck occurs depends on the particular services and how the site uses them. Although the Web server itself may be able to handle a very large number of user sessions, other services within the Web site may balk when asked to service more requests than they can handle at once.

While a Web site may use databases, file services, and other server-side applications, it may not always, for example, use a database within each session or within a particular part of a transaction within a session. The known limits of concurrency by each of the components the Web site makes use of, however, can be used as part of an estimate of how well the site will scale in relation to user sessions.

One reason determining where the actual problem lies in Web site testing of concurrency, and performance in general, is that besides multiple software components in use on one server-side machine, there may be multiple server-side machines that handle different types of requests. The Web server may attempt to balance load between multiple machines running the same component, there may be separate machines devoted to different types of Web site component (a Web server, an Application Server, and a Database Server, for example), and some combination of specialization and load balancing may also be used.

Results of this test

Problems with multiple sessions by the same user will present themselves as denial of service when that user attempts to start a new session, either on the same client or on a different client machine. The inability of the Web site to handle a particular number of sessions can result in a refusal to handle a new request or, especially if the requests are queued, may result in additional time to service new requests. If the Web site is being tested for performance, longer request handling time may violate expected session service times, appearing as expected connections dropped due to the site's inability to handle sessions with predicted user site abandonment times.

Running this test

To test concurrency with the same user, start up multiple sessions on the same machine and on different machines using the same user ID. To test the ability

of the Web site to handle multiple sessions in general, employ performance-testing tools such as EasyWebLoad (http://www.easywebload.com/), Apache JMeter (http://jakarta.apache.org/jmeter/), Loadrunner and Astra LoadTest, from Mercury Interactive (http://www-svca.mercuryinteractive.com/products/), http_load, from Acme Labs (http://www.acme.com/software/), e-load Expert, from Empirix (http://www.empirix.com/), or one of the many other load and concurrency testing tools available. To find the best tool for the particular Web site under test, it is often helpful to characterize the site as either "high complexity, low traffic", "low complexity, high traffic", or "high complexity, high traffic" (which may prove relatively much harder to test than the other two). [14]

5.3 Test Number 16: Test Stress

Background for this test

Load testing has been defined to have as its focus "the intention of measuring a break in functionality based on variable input (i.e. the number of attachments) but often with the amounts of data remaining constant." The focus of stress testing, however, is "a break or degradation of performance at a point in time based on a constant input (i.e., a constant number of attachments) but often varying the amounts or intensity of the data." [26]

The major difference, then between load testing and stress testing can be thought of as stress testing is load testing taken to an extreme. [27] In Web site testing terms, stress testing is applying a sufficiently high enough request level to observe what happens to the site when it can't keep up with the level of requests.

Rationale for this test

What causes stress to a Web application? Amazingly, the same things that cause stress to people: loss and change. When people experience the loss of loved

ones, livelihood, material possessions, or thesis defenses, or when changes (loss of control of one's environmental resources) are involuntarily thrust upon people, stress occurs. Although each person's strategy for handling stress may differ, each person is responsible for choosing their own course of action and thereby how the losses and changes they experience will alter their lives and the lives of those around them.

When a Web application, just like any other type of software application, experiences stress, it is a result of the loss or change of something the application depends upon for execution because the demand on the service is higher than the service can handle. Possible losses include unavailability of a service such as a database, a credit card verification service, a search engine, a stock market reporting service, a Web server, an XML server, an application server, a legacy program, or any other service the application uses, as well as any type of resource on any system on the Web site. Resources like memory, files, and command processing can bog down and become unavailable to a Web application. How the Web application handles stress is what the tester is investigating.

Results of this test

The Web application may ignore the problem, continuing on its merry way, processing inputs and producing outputs. Sooner rather than later, this approach will invariably result in unappreciated actions or inactions. For example, a Web application that depends on a database may fail to record data that might be useful later on, or it might react in one of several wildly unpredictable ways simply because data that would have directed its course of action was not available and its unavailability was ignored.

Running this test

To test stress on a Web application, deprive it of one or more services; give it a "loss" to deal with. The application may remain oblivious to the loss, it may

attempt to partially recover from the loss, or it may refuse to continue performing the work that requires that service or resource until it is restored. The other important piece of data the tester needs to note is what sort of informative or error messages the Web application gave the user when the loss occurred. (Of course, if the loss was not directly detected, the resulting course of action is likely to result in informative or error messages that do not truly reflect the original deprivation.)

The tester can use a software test tool such as Holodeck, available from FIT's Center for Information Assurance (**http://www.se.fit.edu/**) on a machine on the server side of a Web site to simulate low system resources or loss of services. Several performance-testing tools also purport to be able to perform stress testing on Web sites. These tools include EasyWebLoad (**http://www.easywebload.com/**), Loadrunner and Astra LoadTest, from Mercury Interactive (**http://www-svca.mercuryinteractive.com/products/**), and e-load Expert, from Empirix (**http://www.empirix.com/**).

References

1. Mike Powers, *Why Test the Web? How Much Should You Test?* , Testers' Network, January 2000, Available: **http://www.veritest.com/testers'network/Web_testing21.asp**, Last accessed: February 20, 2002.

2. Eric Kaufman, *Testing Your Web Site*, Testers' Network, November 1999, Available: **http://www.veritest.com/testers'network/Web_testing1-1.asp**, Last accessed: February 20, 2002.

3. Daniel J. Mosley, Client-Server Software Testing on the Desktop and the Web, Prentice Hall PTR, 2000.

4. The Word Spy, *Web rage*, Logophilia Website, November 15, 2000, Available: **http://www.logophilia.com/WordSpy/Webrage.asp**, Last accessed: March 12, 2002.

5. Jonathan Lehrer, *Web Rage – User Frustrations of the Internet*, The Web Audit Group, Available: **http://www.webauditgroup.com/advice/frustrate.shtml**, Last accessed: February 20, 2002.

6. James A. Whittaker, How to Break Software, Wiley, 2002.

7. Jesper Rydén and Pär Svensson, Web Application Testing, Chalmers and Sigma nBit AB, Gothenburg, February 2001.

8. Hung Q. Nguyen, Testing Applications on the Web, Wiley, 2001.

9. Dr. Frederick P. Brooks, Jr., The Mythical Man-Month, Essays on Software Engineering, Anniversary Edition, and Addison Wesley Longman, 1995.

10. James Bach, *Testing Internet Software*, Testers' Network, Available: **http://www.veritest.com/testers'network/Inet1.asp**, Last accessed: March 17, 2002.

11. Diane Stottlemyer, Automated Web Testing Toolkit, Wiley, 2001.

12. Steven Splaine and Stefan P. Jaskiel, The Web Testing Handbook, STQE Publishing, 2001.

13. Robert Zemeckis and Bob Gale, *Back to the Future*, Universal City Studios, Inc., 1985.

14. Mark D. Anderson, *The Top 13 Mistakes in Load Testing Applications*, STQE Magazine, Available: http://www.stickyminds.com, Last accessed June 20, 2002.

15. Alberto Savoia, *Trade Secrets from a Web Testing Expert*, STQE Magazine, Available: http://www.stickyminds.com, Last accessed June 20, 2002.

16. Glenn A. Stout, *Testing a Website: Best Practices*, The Revere Group, Available: http://www.reveregroup.com/exchange/articles/stout2.pdf, April 2001, Last Accessed: June 20, 2002.

17. Gerry Ocampo, Testing Considerations for Web-Enabled Applications, Testers' Network, September 1999, Available: **http://www.veritest.com/testers'network/testing_considerations1.asp**, Last accessed: February 20, 2002.

18. Hyoung-Rae Kim, Personal Communication, April 19, 2002.

19. A. J. Flavell, *Use of alt Texts in imgs*, Glasgow University, 1994, Available: http://www.ppewww.ph.gla.ac.uk/~flavell/alt/alt-text.html, Last accessed: October 20, 2002.

20. Frontend, *Website Accessibility*, Frontend Research, March 11, 2001, Available: http://infocentre.frontend.com/servlet/Infocentre/Infocentre?page=article&id=74, Last accessed October 20, 2002.

21. Andrew Chak, *Usability Tools: A Useful Start*, New Architect, 2002, Available: http://www.webtechniques.com/archives/2000/08/stratrevu/, Last accessed: October 20, 2002.

22. NIST, *NIST Web Metrics -- Technical Overview*, NIST Web site, 2002, Available: http://zing.ncsl.nist.gov/WebTools/tech.html, Last accessed: October 20, 2002.

23. Jean Kaiser, *Criteria for Web Site Evaluation*, CNET.com Web site, Available: http://webdesign.about.com/library/weekly/aa071801a.htm, Last accessed: October 20, 2002.

24. Anne Chen, *Web Services Secure?*, eWeek, May 27, 2002, pp. 47-50.

25. Russ Smith, *Behind Closed Doors – What Every Tester Should Know About Web Privacy*, STQE Magazine, January/February 2001, Available: http://www.stickyminds.com/getfile.asp?ot=XML&id=5064&fn=Smzr1XDD2554filelistfilename1%2Epdf, Last accessed October 20, 2002.

26. Akil H. Azizi, *Defining Load Testing*, Testers' Network, Available: http://www.veritest.com/tester%27network/loadtest1.asp, Last accessed: February 20, 2002.

27. Matt Baskett, *Stress and Load Testing on the Web*, Testers' Network, Available: http://www.veritest.com/tester%27network/stressWeb1.asp, Last accessed: February 20, 2002.

28. W3C, *About the Word Wide Web*, W3C, January 24, 2001, Available: http://www.w3.org/WWW/, Last accessed: March 19, 2002

Additional Resources

1. Louise Tamres, Introducing Software Testing, Addison Wesley, 2002.

2. Ron Patton, Software Testing, SAMS Publishing, 2001.

3. Cem Kaner, Jack Falk, Hung Quoc Nguyen, Testing Computer Software, Wiley, 1999.

4. Stefan Asböck, Load Testing for Confidence, Segue Software, Inc., 2000.

5. Segue Software, Inc., *Gain Econfidence: The E-Business Reliability Survival Guide*, Segue Software, Inc., 2000.

6. Star East 2002 Conference Proceedings, SQE, 2002.

7. Jennifer DeJong, *Software Testing: The Internet Changes Everything*, Software Development Times, April 1, 2002, Available: http://www.sdtimes.com/news/051/special1.htm, Last accessed: May 31, 2002.

8. Robert L. Glass, *Has Web Development Changed the Meaning of Testing?* , StickyMinds.com Website, Available: http://www.stickyminds.com/pop_print.asp?ObjectId=2163&ObjectType=ARTCOL, Last accessed: June 18, 2002.

9. James A. Whittaker, *Software's Invisible Users*, IEEE Software, May/June 2001.

10. Edward Hieatt and Robert Mee, *Going Faster: Testing the Web Application*, IEEE Software, March/April 2002.

11. Wing Lam, Testing E-Commerce Systems: A Practical Guide, IEEE IT Pro, March/April 2001.

12. Steve Driscoll, *Systematic Testing of WWW Applications*, WebART Web Site, Available: http://www.oclc.org/webart/paper2, Last accessed February 20, 2002.

13. Mark Cundy, *Testing Mobile Applications is Different From Testing Traditional Applications*, Testers' Network, Available: http://www.veritest.com/testers%27Network/mobilevstraditional.asp, Last accessed February 20, 2002.

14. Filippo Ricca and Paolo Tonella, *Analysis and Testing of Web Applications*, IEEE, 2001.

15. Rhonda Dibachi, *Testing e-Commerce: Reducing Your Company's Risk of Doing Business on the Web*, STQE Magazine, http://www.stqemagazine.com, March/April 2001.

16. Edward Miller, *WebSite Testing*, Software Research, Inc., 2000.

17. Hung Q. Nguyen, *Testing Web-Based Applications*, STQE Magazine, http://www.stqemagazine.com, May/June 2000.

18. QA Labs, Inc., "The Living Creature" – Testing Web Applications, QA Labs, Inc., 2000.

19. Paul Reeser and Rema Hariharan, Analytic Model of Web Servers in Distributed Environments, WOSP 2000, ACM, 2000.

20. Marjorie Lovatt, Herding Cats: A Case Study on the Development of Internet and Intranet Strategies within an Engineering Organization, SIGCPR 97, ACM, 1997.

21. Hal Berghei, *HTML Compliance and the Return of the Test Pattern*, Communications of the ACM, Volume 39, Number 2, February 1996.

22. Pamela B. Lawhead and Kathryn F. Gates, *Managing the Development of a Web-Based Project*, Integrating Tech. Into C.S.E 6/96, ACM, 1996.

23. Chaitanya Kallepalli and Jeff Tian, *Measuring and Modeling Usage and Reliability for Statistical Web Testing*, IEEE Transactions on Software Engineering, Volume 27, Number 11, November 2001.

24. Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti, *Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach*, IEEE Transactions on Parallel and Distributed Systems, Volume 13, Number 1, January 2002.

25. Amdreas Zeller and Ralf Hildebrandt, *Simplifying and Isolating Failure-Inducing Input*, IEEE Transactions on Software Engineering, Volume 28, Number 2, February 2002.

26. Brian Globerman, *Online Retailing Makes Its Mark*, Testers' Network, Available: http://www.veritest.com/tester%27network/online_retailing1.asp, Last accessed: February 20, 2002.

27. Karen Johnston, *HTML Validation*, Testers' Network, Available: http://www.veritest.com/tester%27network/htmlvalid1.asp, Last accessed: February 20, 2002.

28. Richard Brauchle, *How to Test Cookies in a Stateful Web System*, StickyMinds.com Website, Available: http://stickyminds.com/swtest.asp?zone=WBTST&sid=1511405&sqry=%2AJ%28ARTCOL%29%2AR%28createdate%29%2AK%28topicarea%29%2AA%28WBTST%29%2A&sidx=18&sopp=10&ObjectId=2935&Function=DETAILBROWSE&ObjectType=ART, Last accessed: June 18, 2002.

29. Melody Y. Ivory and Marti A. Hearst, *The State of the Art in Automating Usability Evaluation of User Interfaces*, ACM Computing Surveys, Volume 33, Number 4, December 2001.

30. Dion Johnson, *Designing an Automated Web Test Environment*, Pointe Technology Group, Inc., May 17, 2001.

31. Ray Robinson, *Automation Test Tools*, mailto:ray.robinson@cableinet.co.uk, September 11, 2001.