Testing Web Applications

Hanan Qassim Jaleel
Baghdad College of Medical Sciences

Abstract

Web applications are meant to be viewed by human user. While this implies that quality of web application has importance in our daily life. Web application quality is our prime concern. To ensure the quality of web application, web testing is having a dandy role in Software Testing as well as Web Community. Web Applications are erring because of features provided for rising of web application. In the last years, various web testing problems have been addressed by research work. Several tools, techniques and methods have been determined to test web application efficaciously. This paper will present the testing types , testing approaches and methods that can be used to achieve web testing

I. Introduction

Web application testing is a software testing technique exclusively adopted to test the applications that are hosted on web in which the application interfaces and other functionalities are tested.

Web applications are the fastest growing classes of software systems today. Web applications are being used to support wide range of important activities: business transaction, scientific activities like information sharing, and medical systems such as expert system-based diagnoses.

Some errors are the result of incorrect design or improper coding(in HTML , client side scripting , server side programming), other errors due to the static operating environment (the specific configuration in which testing is conducted) , others due to dynamic operating envirnments.

In fact, because Web-based systems and applications reside on a network and interoperate with many different operating systems, browsers [or other interface devices such as mobile phones], hardware platforms, communications protocols, the search for errors represents a significant challenge for Web engineers[1].

A. Web application can be considered as a distributed system including the following main characteristics:

1) A wide number of users distributed all over the world and accessing it concurrently.
2) Heterogeneous execution environments composed of different hardware, network connections, operating systems, Web servers and Web browsers.
3) An extremely heterogeneous nature that depends on the large variety of software components that it usually includes. These components can be constructed of different technologies (i.e., different programming languages and models).
4) The ability of generating software components at run time according to user inputs and server status[1,2].

II. Testing Types

There are several types of testing, these are:

A. Content Testing

Faults in WebApp content can be as trivial as minor typographical errors or as significant as incorrect information, improper organization, or violation of intellectual property laws. Content testing attempts to uncover these errors and many other problems before the user encounters them.

Content testing has three important objectives:

a) To uncover syntactic errors (e.g., typos, grammar mistakes) in text-based documents, graphical representations, and other media.
b) To uncover semantic errors (i.e., errors in the accuracy or completeness of information).
c) To find errors in the organization or structure of content that is presented to the end user[1,3].

To accomplish the first objective, **automated spelling and grammar checkers** may be used. However, many syntactic errors evade detection by such tools and must be discovered by a human reviewer (tester). Semantic testing focuses on the information presented within each content object.

- ***DataBase Testing***

Modern WebApps do much more than present static content objects. In many application domains, WebApps interface with sophisticated database management systems and build dynamic content objects that are created in real time using the data acquired from a database.

To accomplish this, the following steps are required:

(1) a large database is queried,

(2) relevant data are extracted from the database,

(3) the extracted data must be organized as a content object, and

(4) this content object (representing customized information requested by an enduser) is transmitted to the client environment for display. Errors can and do occur as a consequence of each of these steps.

The objective of database testing is to uncover these errors. Testing should ensure that

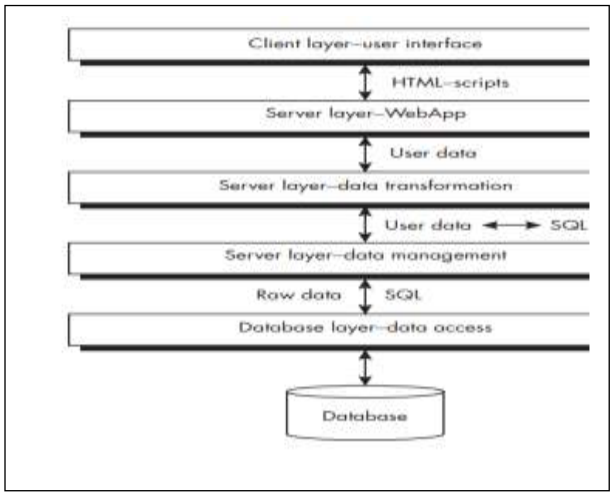
(1) valid information is passed between the client and server from the interface layer,

(2) theWebApp processes scripts correctly and properly extracts or formats user data,

(3) user data are passed correctly to a server-side data transformation function that formats appropriate queries (e.g., SQL), and

(4) queries are passed to the data management layer 9 that communicates with database access routines (potentially located on another machine)[1,4].

Data transformation, data management, and the database access layers shown in Figure 1


Client layer–user interface
HTML–scripts
Server layer–WebApp
User data
Server layer–data transformation
User data
SQL
Server layer–data management
Raw data
SQL
Database layer–data access
Database

Figure(1) layers of interaction

B. User Interface Testing

The overall Objectives for interface testing is to:

(1) uncover errors related to specifi c interface mechanisms (e.g., errors in the proper execution of a menu link or the way data are entered in a form), and

(2) uncover errors in the way the interface implements the semantics of navigation, WebApp functionality, or content display[1].

Interface Mechanisms

When a user interacts with a WebApp, the interaction occurs through one or more interface mechanisms. These mechanisms include :

Testing of Interface Mechanisms

- ***Links***. Each navigation link is tested to ensure that the proper content object or function is reached. You can build a list of all links associated with the interface layout (e.g., menu bars, index items) and then execute each individually. In addition, links within each content object must be exercised to uncover bad URLs or links to improper content objects or functions. Finally, links to external WebApps should be tested for accuracy and also evaluated to determine the risk that they will become invalid over time[5].

- ***Forms***

(1) labels correctly identify fields within the form and that mandatory fields are identified visually for the user,
(2) the server receives all information contained within the form and that no data are lost in the transmission between client and server,

(3) appropriate defaults are used when the user does not select from a pull-down menu or set of buttons,

(4) browser functions (e.g., the back arrow) do not corrupt data entered in a form,
(5) scripts that perform error checking on entered data work properly and provide meaningful error messages
(6) form fields have the proper width and data types,
(7) the form establishes appropriate safeguards that preclude the user from entering text strings longer than some predefined maximum,
(8) all appropriate options for pull-down menus are specified and ordered in a way that is meaningful to the end user,
(9) browser “auto-fill” features do not lead to data input errors,

(10) the tab key (or some other key) initiates proper movement between form fields[1,5].

- ***Client-side scripting***.tests are conducted to uncover any errors in processing as the script is executed. These tests are often coupled with forms testing, because script input is often derived from data provided as part of forms processing.

- ***Dynamic HTML***. Each Web page that contains dynamic HTML is executed to ensure that the dynamic display is correct.

- ***Pop-up windows***. A series of tests ensure that:

  (1) the pop-up is properly sized and positioned,
  (2) the pop-up does not cover the original WebApp window,
  (3) the aesthetic design of the pop-up is consistent with the aesthetic design of the interface, and
  (4) scroll bars and other control mechanisms appended to the pop-up are properly located and function as required

- ***Server-side scripts***. tests are conducted with an emphasis on data integrity (as data are passed to the Server-side script) and script processing once validated data have been received.

- ***Streaming content***. Tests should demonstrate that streaming data are up to date, properly displayed, can be suspended without error, and can be restarted without difficulty

- ***Cookies***. Both server-side and client-side testing are required. On the server side, tests should ensure that a cookie is properly constructed (contains correct data) and properly transmitted to the client side when specific content or functionality is requested.. On the client side, tests determine whether the WebApp properly attaches existing cookies to a specific request (sent to the server).

Interface semantics testing "evaluates how well the design takes care of users, offers clear direction, delivers feedback, and maintains consistency of language and approach"[1,2,5].

C. Usability Testing

Usability testing is similar to interface semantics testing , it also evaluates the degree to which users can interact effectively with the WebApp and the degree to which the WebApp guides users' actions, provides meaningful feedback, and enforces a consistent interaction approach.

Usability testing can occur at a variety of different levels of abstraction:

(1) the usability of a specific interface mechanism (e.g., a form) can be assessed,

(2) the usability of a complete Web page (encompassing interface mechanisms, data objects, and related functions) can be evaluated,

(3) the usability of the complete WebApp can be considered.

The following test categories and objectives illustrate this approach:

Interactivity. The interaction mechanisms (e.g., pull-down menus, buttons, pointers) should be easy to understand and use

Layout. The navigation mechanisms, content, and functions should be placed in a manner that allows the user to find them quickly

Readability. The text should be well written and understandable , and the graphic representations should be intuitive and easy to understand

Aesthetics. the layout, color, typeface, and related characteristics should lead to ease of use , and users are "feel comfortable" with the look and feel of the WebApp
Display characteristics. theWebAppshould make the optimal use of screen size and resolution

Time sensitivity.the important features, functions, and content should be used or acquired in a timely manner

Personalization. theWebApp appropriately tailor itself to the specific needs of different user categories or individual users

Accessibility.theWebApp should be accessible to people with disabilities

the following interface features should be reviewed and tested for usability: animation, buttons, color, control, dialogue, fields, forms, frames, graphics, labels, links, menus, messages, navigation, pages, selectors, text, and tool bars[1,6]

D. Compatibilty Testing

- WebApps must operate within environments that differ from one another. Different computers, display devices, operating systems, browsers, and network connection speeds can have a significant influence on WebApp operation.
- The most common problem arising from compatibility issues is poor usability. This can arise from the following: download speeds may become unacceptable, lack of a required plug-in may make content unavailable, browser differences can change page layout dramatically, font styles may be altered and become illegible, or forms may be improperly organized. **Compatibility testing strives to uncover these problems before the WebApp goes online[2,7]**

E. Component Level Testing

Component-level testing, also called function testing, focuses on a set of tests that attempt to uncover errors in WebApp functions. Each WebApp function is a software module (implemented in one of a variety of programming or scripting languages) and can be tested using different techniques. Component-level test casesare often driven by forms-level input. Once forms data are defined, the user selects a button or othercontrol mechanism to initiate execution. **The following test case design methods are typical:**

- ***Equivalence partitioning***. The input domain of the function is divided into input categories or classes from which test cases are derived. Test cases for each class of input are derived and executed, while other classes of input are held constant.

For example, an e-commerce application may implement a function that computes shipping charges. Among a variety of shipping information provided via a form is the user's postal code. Test cases are designed in an attempt to uncover errors in postal code processing by specifying postal code values that might uncover different classes of errors (e.g., an incomplete postal code, a correct postal code, a nonexistent postal code, an erroneous postal code format).errors (e.g., an incomplete postal code, a correct postal code, a nonexistent postal code, an erroneous postal code format)[2,8].

- ***Boundary value analysis***. Forms data are tested at their boundaries. ***For example***, a shipping calculation function requests the maximum number of days required for product delivery. A minimum of 2 days and a maximum of 14 are noted on the form. However, boundary value tests might input values of 0, 1, 2, 13, 14, and 15 to determine how the function (and associated error processing) reacts to data at and outside the boundaries of valid input.

F. Navigation Testing

A user travels through a WebApp in much the same way as a visitor walks through a store or museum. There are many pathways that can be taken, many stops that can be made, many things to learn and look at, activities to initiate, and decisions to make. This navigation process is predictable in the sense that visitors have a set of objectives when they arrive. At the same time, the navigation process can be unpredictable because visitors, influenced by something they see or learn, may choose a path or initiate an action that is not typical for the original objective.

Each of the following navigation mechanisms should be tested

- ***Navigation links.*** These mechanisms include internal links within the WebApp, external links to other WebApps, and anchors within a specif c Web page. Each link should be tested to ensure that proper content or functionality is reached when the link is chosen
- ***Redirects***. These links come into play when a user requests a nonexistent URL or selects a link whose destination has been removed or whose name has changed. A message is displayed for the user, and navigation is redirected to another page (e.g., the home page).
- ***Site maps.*** A site map provides a complete table of contents for all Web pages. Each site map entry should be tested to ensure that the link takes the user to the proper content or functionality.
- ***Internal search engines.*** Complex WebApps often contain hundreds or even thousands of content objects. An internal (local) search engine allows the user to perform a key word search within the WebApp to find needed content. Search engine testing validates the accuracy and completeness of the search, the error-handling properties of the search engine, and advanced search features (e.g., the use of Boolean operators in the search field)[1,3,9].

G. Configuration Testing

Configuration variability and instability are important factors that make Web engineering a challenge. Hardware, operating system(s), browsers, storage capacity, network communication speeds, and a variety of other client-side factors are difficult to predict for each user. In addition, the configuration for a given user can change [e.g., operating system (OS) updates, new Internet service provider (ISP), and connection speeds] on a regular basis. The result can be a client-side environment that is prone to errors that are both subtle and significant. One user's impression of the WebApp and the manner in which that user interacts with it can differ significantly from another user's experience, if both users are not working within the same client-side configuration.

The job of configuration testing is not to exercise every possible client-side configuration. Rather, it is to test a set of probable client-side and server-side configurations to ensure that the user experience will be the same on all of them and to isolate errors that may be specific to a particular configuration.

On the server side, configuration test cases are designed to verify that the projected server configuration [i.e., WebApp server, database server, operating system(s), firewall software, concurrent applications] can support the WebApp without error. In essence, the WebApp is installed within the server-side environment and tested to uncover errors as it operates[4,7].

On the client side, configuration tests focus more heavily on WebApp compatibility with configurations that contain one or more permutations of the following components :

• ***Hardware***. CPU, memory, storage, and printing devices

• ***Operating systems***. Linux, Macintosh OS, Microsoft Windows, a mobilebased OS

• ***Browser software***.FireFox, Internet Explorer, Safari, Mozilla/Netscape, Opera, and others

• ***User interface components***. Active X, Java applets, and others

• ***Plug-ins***. QuickTime, RealPlayer, and many others

•***Connectivity***. Cable, DSL, regular modem, industry-grade connectivity (e.g., T1 lines)[10]

H. Security Testing

Security testing focuses on unauthorized access to WebApp content and functionality along with other systems that cooperate with the WebApp on the server side.

Security tests are designed to probe vulnerabilities of the client-side environment, the network communications that occur as data are passed from client to server and back again, and the server-side environment. Each of these domains can be attacked, and it is the job of the security tester to uncover weaknesses that can be exploited by those with the intent to do so.

On the client side, vulnerabilities can often be traced to preexisting bugs in browsers, e-mail programs, or communication software. There are typicalsecurity holes:

- Buffer Overflow
- Unauthorized access to cookies placed within the browser

On the server side, vulnerabilities include denial-of-service attacks and malicious scripts that can be passed along to the client side or used to disable server operations. In addition, server-side databases can be accessed without authorization (data theft)[1,10].

To protect against these (and many other) vulnerabilities, one or more of the following security elements is implemented

• ***Firewalls***

A filtering mechanism that is a combination of hardware and software that examines each incoming packet of information to ensure that it is coming from a legitimate source, blocking any data that are suspect.

• ***Authentication***. A verification mechanism that validates the identity of all clients and servers, allowing communication to occur only when both sides are verified.

• ***Encryption***. An encoding mechanism that protects sensitive data by modifying it in a way that makes it impossible to read by those with malicious intent. Encryption is strengthened by using digital certificates that allow the client to verify the destination to which the data are transmitted.

• ***Authorization.*** A filtering mechanism that allows access to the client or server environment only by those individuals with appropriate authorization codes (e.g., user ID and password)[2].

I. Peformance Testing

Performance testing is used to uncover performance problems that can result from

- lack of server-side resources
- inappropriate network bandwidth
- inadequate database capabilities
- faulty or weak operating system capabilities
- poorly designed WebApp functionality,
- other hardware or software issues that can lead to degraded client-server performance.

The intent of performance testing are

- To understand how the system responds as loading increases(i.e. the number of users , number of transactions , overall data volume)
- To collect metrics that will lead to design modifications to improve performance[11]

Two Different performance tests are conducted:

a) ***Load testing* -** It is the simplest form of testing conducted to understand the behavior of the system under a specific load.
b) ***Stress testing* -** It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum[11].

Load Testing

The intent of load testing is to determine how the WebApp and its server-side environment will respond to various loading conditions. As testing proceeds, permutations to the following variables define a set of test conditions:

N, the number of concurrent users

T, the number of online transactions per unit of time

D, the data load processed by the server per transaction

Overall throughput P is computed in the following manner:

$P = N * T * D$

As an example, consider a popular sports news site. At any given time, 4000 concurrent users submit a request (a transaction T) once every 30 seconds on average. Each transaction requires the WebApp to download a news article that averages 12 kbytes in length. Therefore,

N = 4000 users

T = 0.033 transactions per second per user

D = 12 kbyte per transaction

And throughput can be calculated as

$P = 4000 * 0.033 * 12 \approx 1600$ kbyte/s

The network connection for the server would therefore have to support this average data rate and should be tested to ensure that it does[1,11].

Stress testing

Stress testing is a continuation of load testing, but in this instance the variables, N, T, and D are forced to meet and then exceed operational limits. The intent of these tests is to answer each of the following questions:

Does the system degrade "gently" or does the server shut down as capacity is exceeded?

• Does server software generate "server not available" messages? More generally, are users aware that they cannot reach the server?

• Does the server queue requests for resources and empty the queue once capacity demands diminish? • Are transactions lost as capacity is exceeded?

• Is data integrity affected as capacity is exceeded?

• What values of N, T, and D force the server environment to fail? How does failure manifest itself? Are automated notifications sent to technical support staff at the server site?

• If the system does fail, how long will it take to come back online?

• Are certain WebApp functions (e.g., compute intensive functionality, data streaming capabilities) discontinued as capacity reaches the 80 or 90 percentpercent level?

III. Functional Testing

Testing the functionality of a Web application has to rely on the following basic aspects:

1) ***Test models***: representing the relationships between elements of a representation or an implementation of a software component. Examples are UML, finite state machine; relational management data model (RMDM), and object oriented hypermedia (OOH).

2) ***Test strategies***: define heuristics or algorithms to create test cases from software representation models, implementation models or test models. Examples are white-box testing, black-box testing, and gray-box testing.
3) ***Testing levels****:* specifying the different scopes of the tests to be run, i.e., the collections of components to be tested.
4) ***Testing processes***: define the flow of testing activities, and other decisions regarding when testing should be started, who should perform testing, how much effort should be used and similar issues[2,12].

IV. Testing Approaches

Approaches to Testing

• Black Box ,White Box and Grey Box

• Alpha and Beta

- ***White Box testing***

White box testing is testing where using the information is available from the code of the component to generate tests. This info is usually used to achieve coverage in one way or another – e.g.

• Code coverage • Path coverage • Decision coverage

Debugging will always be white-box testing ,The need of White Box Testing? To discover the following types of bugs:

- Logical error tend to creep into our work when we design and implement functions, conditions or controls that are out of the program
- The design errors due to difference between logical flow of the program and the actual implementation
- Typographical errors and syntax checking[2]

- ***Black Box testing***

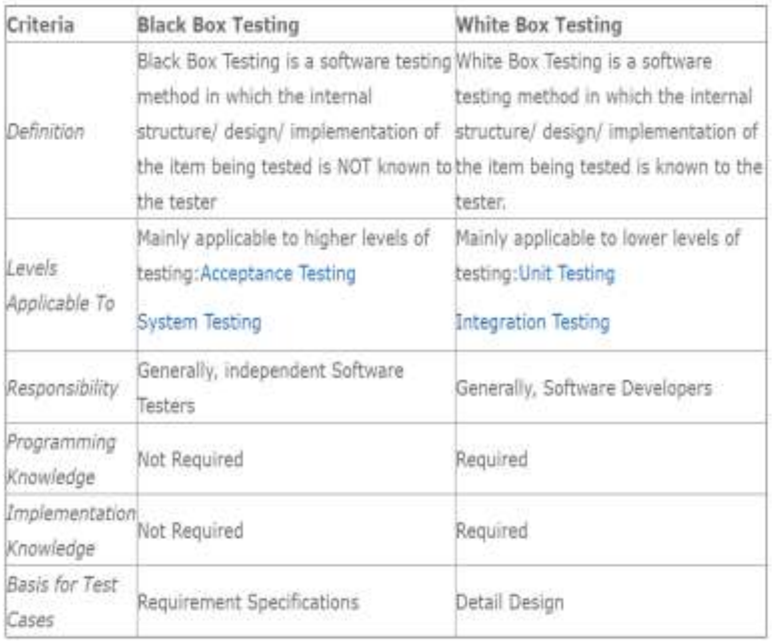
Black box testing is also called functional testing. Main focus in black box testing is on functionality of the system as a whole. The term ‘**behavioral testing’** is also used for black box testing and white box testing is also sometimes called **‘structural testing’**Themain ideas are simple:

1. Define initial component state, input and expectedoutput for the test.

2. Set the component in the required state.

3. Give the defined input

4. Observe the output and compare to the expectedoutput[2,3].



Criteria	Black Box Testing	White Box Testing
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing:Acceptance Testing System Testing	Mainly applicable to lower levels of testing:Unit Testing Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design

Gray box testing
is a software testing method which is a combination of Black Box Testing method and White Box Testing method. In Black Box Testing, the internal structure of the item being tested is unknown to the tester and in White Box Testing the internal structure is known. In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. Gray Box Testing is named so because the software program, in the eyes of the tester is like a gray/semi-transparent box; inside which one can partially see[4,5].

Example
when the codes for two units/modules are studied (White Box Testing method) for designing test cases and actual tests are conducted using the exposed interfaces (Black Box Testing method).
Levels Applicable To

Though Gray Box Testing method may be used in other levels of testing, it is primarily used in **Integration Testing1,2]**.

– Alpha testing: done in-house.
– Beta testing: done at the customer site.
Typical goals of beta testing: to determine if the product worksand is free of "bugs."

Alpha testing and Beta testing (Product Use Testing

Product use under normal operating conditions[1,2,10].

Alpha Testing	Beta Testing (Field Testing)
It is always performed by the developers at the software development site.	. It is always performed by the customers at their own site.
Sometimes it is also performed by Independent Testing Team.	It is not performed by Independent Testing Team.
Alpha Testing is not open to the market and public	Beta Testing is always open to the market and public.
It is conducted for the software application and project.	It is usually conducted for software product.
It is always performed in Virtual Environment.	It is performed in Real Time Environment.
It is always performed within the organization.	It is always performed outside the rganization.
It is the form of Acceptance Testing.	It is also the form of Acceptance Testing.
Alpha Testing is definitely performed and carried out at the developing organizations location with the involvement of developers.	Beta Testing (field testing) is performed and carried out by users or you can say people at their own locations and site using customer data.
It comes under the category of both White Box Testing and Black Box Testing.	It is only a kind of Black Box Testing.
. Alpha Testing is always performed at the time of Acceptance Testing when developers test the product and project to check whether it meets the user requirements or not.	. Beta Testing is always performed at the time when software product and project are marketed.
It is always performed at the developer's premises in the absence of the users.	It is always performed at the user's premises in the absence of the development team.
Alpha Testing is not known by any other different name.	Beta Testing is also known by the name Field Testing means it is also known as field testing.
It is considered as the User Acceptance Testing (UAT) which is done at developer's area.	It is also considered as the User Acceptance Testing (UAT) which is done at customers or users area.

References

[1] Roger S. Pressman & David Lowe ,"Web Engineering a Practitioner's Approach", McGraw-Hill higher education , 2009.

[2] Arora A. & Sinha M ,"Web Application Testing: A Review on Techniques, Tools and State of Art", International Journal of Scientific & Engineering Research, Volume 3, Issue 2, February-2012

[3] Roopa Singh & Imran Akhtar Khan ,"AN APPROACH FOR INTEGRATION TESTING IN ONLINE RETAIL APPLICATIONS ",International Journal of Computer Science & Information Technology (IJCSIT) Vol 4, No 3, June 2012

[4] Glenn A. Stout ," Testing a Website: Best Practices ",Senior Functional Specialist The Revere Group , August, 2001

[5] FilippoRicca and Paolo Tonella," Analysis and Testing of Web Applications", Italy , IEEE , 2001

[6] Iulia Ştefan and Ioan Ivan ," WEB TESTING APPLICATION WITH PHP AUTOMATED TOOL", Department of Automation, Technical University, Cluj-Napoca, Romania , 2014

[7] Shay Artzi , Julian Dolby , Simon Holm Jensen, Anders Møller , Frank Tip , "A Framework for Automated Testing

of JavaScript Web Applications" , Honolulu, Hawaii, USA , 2011

[8] Franck Lebeau , Bruno Legeard , Fabien Peureux , and Alexandre Vernotte ," Model-Based Vulnerability Testing for Web Applications", France , http://www.dvwa.co.uk/

[9] Regina Ranstrom ,"Automated Web Software Testing With Selenium", Department of Computer Science and Engineering University of Notre Dame, Notre Dame.

[10] Giuseppe A. Di Lucca a & Anna Rita Fasolino," Testing Web-based applications: The state of the art and future trends", August 2006

[11] Krishen Kota, PMP ," Testing Your Web Application A Quick 10-Step Guide ",white paper.

[12] G. Cassone, G. Elia, D. Gotta, F. Molaand A. Pinnola," Web Performance Testing and Measurement: a complete approach" , Italy.