International Association of Scientific Innovation and Research (IASIR)
(An Association Unifying the Sciences, Engineering, and Applied Research)

ISSN (Print): 2279-0063
ISSN (Online): 2279-0071

International Journal of Software and Web Sciences (IJSWS)
www.iasir.net

Finite State Machine based Testing of Web Applications

Kulvinder Singh[1], Semmi[1] and Iqbal Kaur[2]

[1]Department of Computer Science and Engineering, UIET,

Kurukshetra University, Kurukshetra

[2]Govt P.G. College, Karnal, Haryana, India

Abstract:- *Web testing is an effective technique to ensure the quality of Web applications. With the development and implementation of web applications in variety of industries, testing web system becomes more and more important and difficult. This literature review paper addresses the use of practical usage model based on finite state machine. According to the FSM usage model, it details the process of generating test cases. Finite state machine are the best method for developing computational model for hardware and software. By using the concept of finite State modeling, a computational model of web application can be built. In this model, web pages can be considered as states of finite state model and links can be considered as state transitions with input condition provided at each state. After building computational model test cases can be generated by using various techniques [2].*

Keywords: *Web Testing, Web application, Finite state machines, Computational Model, Test Cases*

I. INTRODUCTION

A. *Web Engineering*

Web-based systems and applications (WebApps) bring a complex range of content and functionality to a broad population of end-users. Web engineering is the process used to produce high-quality WebApps. web engineering focuses on the methodologies, techniques and tools that are the foundation of web application development and which support their design, development, evolution, and evaluation. The Web engineering process begins with a formulation of the problem to be solved by the WebApp. The project is planned, and the requirements of the WebApp are analyzed. Architectural, navigational, and interface design are conducted. The system is implemented using specialized languages and tools associated with the Web, and testing commences. Because WebApps develop continuously, mechanisms for configuration control, quality assurance, and ongoing support are needed.

B. *The Testing Process*

Web testing is the process of revealing errors that is used to give confidence that the implementation of a Web application meets its specification. The testing process for Web Engineering begins with tests that exercise content and interface functionality that is immediately visible to end users. As testing proceeds, aspects of the design architecture and navigation are exercised. The user may or may not cognizant of these WebApp elements. Finally, the focus shifts to tests that exercise technological capabilities that are not always apparent to end-user WebApp infrastructure and installation issues [1].

As the testing flow proceeds from left to right and top to bottom, user visible elements of the WebApp design are tested first, followed by infrastructure design elements. We will begin testing process with its content through content testing with the attempt to uncover errors in content. In this activity like copy editing for a written document is tested. [1].It uncovers typographical errors, grammatical mistakes, errors in content consistency, cross referencing errors etc. In this step we not only examine static content but the dynamic content is also is also examined that is derived from data maintained as part of a database system which is integrated with the WebApp.

C. *Content testing*

It attempts to uncover errors in content. It uncovers typographical errors, grammatical mistakes, errors in content consistency, errors in graphical representations, errors in the organization and structure of content that is presented to the end-user. In addition to examining static content for errors, this testing step also considers dynamic content derived from data maintained as part of a database system that has been integrated with the WebApp.

Database Testing: In order to obtain a database system which satisfies the ACID properties (Atomicity, Consistency, Isolation, and Durability) of a database management system, Database testing is done. Check for data integrity and errors while you edit, delete, modify the forms or do any DB related functionality. Check if all the database queries are executing correctly, data is retrieved correctly and also updated correctly.

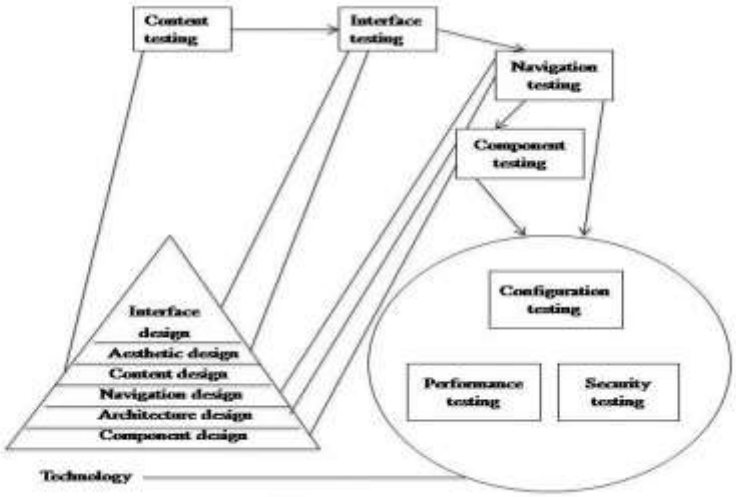

Fig. 1. Testing process

Testing should ensure that:

- The WebApp processes scripts correctly and properly extracts or formats user data.
- User data are passed correctly to a server side data transformation function that formats appropriate queries.
- Queries are passed to a data management layer that communicates with database access routines.
- Valid information is passed between the client and server from interface layer.

The user interface layer is tested to ensure that HTML scripts are properly constructed for each user query and properly transmitted to the server side. The WebApp layer on the server side is tested to ensure that the user data are properly extracted from HTML scripts and properly transmitted to the data transformation layer on the server side. The data transformation functions are tested to ensure that correct SQL is created and passed to appropriate data management components.

D. *User Interface Testing*

Interface testing exercises interaction mechanisms and validates aesthetic aspects of the user interface. The intent is to uncover errors that result from poorly implemented interaction mechanisms, inconsistencies or ambiguities that have been introduced into the interface inadvertently.

1) Testing Interface Mechanisms:

- Links – Each navigation link is tested to ensure that the proper content object or function is reached. The Web engineer builds a list of all links associated with the interface layout (e.g., menu bars, index items) and then executes each individually.
- Forms – tests are performed to ensure that Labels correctly identify fields within the form. The server receives all information contained within the form. Appropriate defaults are used when the user does not select from a pull down menu or set of buttons.

2) *Cookies Testing*: Cookies are small files stored on user machine. These are basically used to maintain the login sessions. Test the application by enabling or disabling the cookies in your browser options. Test if the cookies are encrypted before writing to user machine. On server side, test should ensure that a cookie is properly constructed and properly transmitted to the client side when specific functionality is requested. On the client side, tests determine whether the WebApp properly attaches existing cookies to a specific request.

3) Usability Testing: it 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. The test categories and objectives of usability testing are:

- Interactivity – Are interaction mechanisms (e.g., pull down menus, buttons, pointers) easy to understand?
- Layout – Are navigation mechanisms, content and functions placed in a manner that allows the user to find them quickly?
- Readability – Is text well-written and graphic representation easy to understand?
- Aesthetics – Do users feel comfortable with the look and feel of the WebApp?
- Time sensitivity – Can important features, functions be acquired in a timely manner?

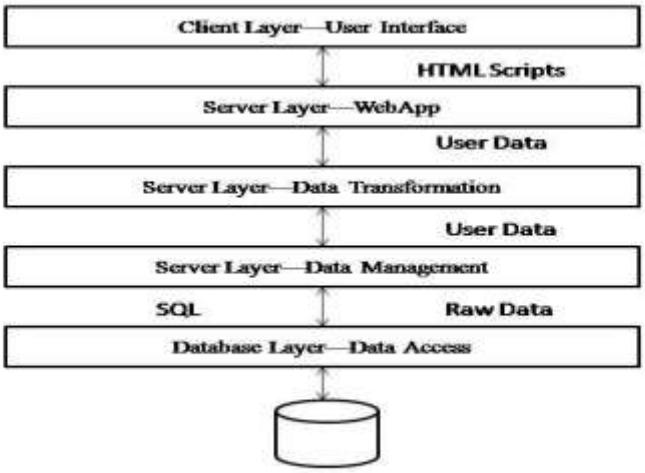

Fig.2. Layers of interaction

E. Navigation testing

It applies use-cases, derived as part of the analysis activity, in the design of test cases that exercise each usage scenario against the navigation design. First phase of navigation testing actually begins during interface testing. Each of the following mechanisms should be tested.

1) *Navigation Links:* Internal links within the WebApp, external links to other WebApps, and anchors within a specific web page should be tested to ensure that proper content or functionality is reached when the link is chosen.
2) *Redirects*: These links come into play when a user requests a nonexistent URL or selects a link whose destination has been removed. A message is displayed for the user, and the navigation is redirected to another page. Redirects should be tested by requesting incorrect internal links or external URLs and assessing how the WebApp handles these requests.
3) *Bookmarks*: The WebApp should be tested to ensure that a meaningful page title can be extracted as the bookmark is created.
4) *Site maps:* Entries should be tested to ensure that the link takes the user to the proper content or functionality.
5) *Internal Search Engines*: Complex WebApps often contains hundreds or even thousands of content objects. An internal search engine allows the user to perform a keyword search within the WebApp to find needed content. Search engine testing validates the accuracy and completeness of the search.

F. Component Level Testing

It exercises content and functional units within the WebApp. The "unit" of choice within the content architecture is the web page. Each web page encapsulates content, navigation links, and processing elements (forms, scripts). Each functional component is tested in much the same way as an individual module is tested in conventional software. Test case design methods for component testing are [1]:

1) *Equivalence Partitioning:* The input domain of the function is divided into 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.
2) *Boundary Value Analysis*: Forms data are tested at their boundaries.

3) *Path Testing*: If the logical complexity of the function is high, path testing can be used to ensure that every independent path in the program has been exercised.

G. Configuration testing

It attempts to uncover errors that are specific to particular client or server environment. A cross-reference matrix is created that defines all feasible operating systems, browsers, hardware platforms, and communication protocols. Tests are then conducted to uncover errors associated with each possible configuration.

1) *Server side issues:* On the sever side, configuration test cases are designed to verify that the projected server configuration (i.e., WebApp server, database server, operating systems, firewall software, concurrent applications) can support the WebApp without error.
2) *Client side issues:* On the client side, configuration test focus more heavily on WebApp compatibility with configurations that contain one or more permutation of the components: Hardware,Operating systems, Browser software, Connectivity.

H. Security testing

It incorporates a series of tests designed to exploit vulnerabilities in the WebApp and its environment. To protect against these vulnerabilities, one or more of the following security elements are implemented:

1) *Firewall*: a filtering 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.
2) *Authentication:* a verification mechanism that validates the identity of all clients and servers, allowing communication to occur only when both sides are verified.
3) *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.
4) *Authorization:* A filtering mechanism that allows access to the client or server environment only by those individuals with appropriate authorization codes (e.g., userID and password).

I. Performance testing

It 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, and other hardware or software issues that can lead to degraded client server performance [1]. Types of Performance Testing are:

1) *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 on-line transactions per user per unit time
   D, the data load processed by the server per transaction
   P=N x T x D
2) *Stress Testing*: Loading is increased to the breaking point to determine how much capacity the WebApp environment can handle.

II. Previous Research

Andrews *et al.* [3] proposed in their paper, a system-level testing technique that combines test generation based on finite state machines with constraints. Researchers used a hierarchical approach to model potentially large web applications. The approach builds Finite State Machines that models subsystems of the web application and then generates test requirements as subsequences of states in the FSMs. These subsequences were then combined and refined to form complete executable tests. The constraints were used to select a reduced set of inputs with the goal of reducing the state space explosion otherwise inherent in using FSMS. The paper illustrates the technique with a running example of a web-based course student information system and introduces a prototype implementation to support the technique.

Song *et al*. [4] proposed Modeling Database Interactions in Web applications and Generating Test Cases. In their paper, special care was taken on database interactions in modeling and testing Web applications. An augmented FSMs (GFSMs) were employed as a tool to model database interactions. This work proposed a Web testing model with database interactions for generating test. It started from constructing the GFSM test-tree derived from FSM of

the Web application. An algorithm is then designed to construct a minimal test set of GFSM test tree. Then test paths were generating to cover each element of test set. Finally, an algorithm was designed to optimize the test paths by decreasing the overlap. The approach we proposed can yield substantial results with test paths and state transitions were all less.

Song *et al.* [5] proposed Model Composition and Generating Tests for Web Applications. Web testing is one of the methods to ensure the Web security. A typical Web application consists of two tiers: the client, the server. Most existing works consider Web applications testing from an external or a user's view, not taken the interactions and the behaviors of the server side into account. In this paper, special care on the interactions and the behaviors of server side was paid, and an approach to model composition and testing Web applications was proposed. FSMs were used to model Web applications from the user's side, and the server's side, respectively. Then, synchronous product was employed as a tool to construct a composition of FSMs. Finally, based on the composition of FSM, test generation is given out which satisfied the corresponding coverage criteria. Web applications were widely used in our daily life. Due to the client interaction or communication with the server by sending message requests and response, the synchronous product is employed to finish the composition of the client model and the server side model. Finally, based on the composition model, the tests were generated.

Hierons *et al.* [6] presented the concept of mutation testing with PFSM. Mutation testing traditionally involves mutating a program in order to produce a set of mutants and using these mutants in order to either estimate the effectiveness of a test suite or to derive test generation. However in their paper, this approach had been applied to specification such as those written as finite state machines. This paper extended mutation testing to finite state machine models in which transitions had associated probabilities. The paper described several ways of mutating a probabilistic finite state machine (PFSM) and showed how test sequences that distinguish between a PFSM and its mutants could be generated. Testing then involved applying each test sequence multiple times, observing the resultant output sequences and using result from statistical sampling theory in order to compare the observed frequency of each output sequence with that expected.

Cavalli *et al*. [7] proposed WebMov: A dedicated framework for the modeling and testing of Web Services Composition. They presented the research work and the experimental results of the project WebMov, whose main objective was to study different methodologies and to develop a set of tools covering all the phases of the development and validation cycle of Web Services composition. This framework also permitted the interaction of these tools to achieve specific modeling and testing activities in a complementary way. The whole WebMov methodology was integrated within a dedicated framework, composed by a set of tools that implemented the model representation, the test generation and passive testing algorithms. As presented in the paper, different formal techniques have been developed for web services modeling based on different variants of Timed Extended Finite State Machines. These techniques had been combined with fault injection in order to perform conformance and robustness testing.

Kalaji *et al*. [8] mentioned that the problem of testing from an extended finite state machine model (EFSM) can be expressed in terms of finding suitable paths through the EFSM and then driving test data to follow the paths. A chosen path may be infeasible and so it is desirable to have methods that can direct the search for appropriate paths from the EFSM towards those that are likely to be feasible. However, generating feasible transition paths (FTPs) for the model based testing is a challenging task and is an open research problem. This paper introduces a novel fitness metric that analyzes data flow dependence among the actions and conditions of the transitions of a path in order to estimate its feasibility. The proposed fitness metric is evaluated by being used in a genetic algorithm to guide the search for FTPs. The approach had two phases: the aim of the first phase being to produce a feasible TP (FTP) while the second phase searches for an input sequences to trigger this TP. The path input sequence generation problem is to find an input sequence that can traverse a feasible path. This second phase uses a genetic algorithm whose fitness function is based on a combination of a branch distance function and approach level [9].

Wang *et al*. [10] proposed An Approach to Transform UML Model to FSM Model for Automatic Testing. As they focused on the translation of state diagrams, a specific mechanism was proposed which enables generation of FSM models with same semantics. Modelers created one state diagram for each object and other diagrams for relations between them. Among various modeling languages, UML is widely spread and used for its simplicity, understandability and ease of use. But rigorous analysis for UML model is difficult due to its lack of precise semantics. On the other hand, as a formal notation, FSM provides an avenue for automatic generation of test cases. Therefore they described an approach to transforming UML model to FSM model, taking advantage of both

languages. To illustrate the mechanism we proposed, they gave a method for implementation of the mechanism and a tool prototype to support the method.

Liping *et al*. [11] proposed An Approach to Testing Web Applications Based on Probable FSM. According to the probable FSM usage model, they detailed the process of generating test cases. The testing process was based on the idea that different parts of the Web application had different execution frequency. Therefore, the test cases produced ensure that the failures occurring most frequently in operational uses will be found early in the test process. Statistical usage testing was based on the idea that different parts of the software don't need to be tested with the same thoroughness. The primary benefit of statistical testing was that it allows the use of statistical inference techniques to compute probabilistic aspects of the testing process. The approach used in this paper treats Web applications according to their statistical usage models, which divides a Web application into subWAs and each subWA will not raise the state space explosion problem if the Web application under test is divided reasonably. It aimed at statistical reliability testing rather than fault detection only.

III. FUTURE SCOPE AND CONCLUSION

Testing of web application is a very complicated and vast concept. Instead of that there are various techniques available for web application testing. This paper mentioned the techniques of finite state machines. Finite state machine are the best method for developing computational model for hardware and software. Design criteria of finite state machine can be expressed in terms of desirable and undesirable states and state transitions. In this paper a number of research papers have been mentioned on various testing techniques which give the knowledge of different types of testing of the web application. By using the concept of finite State modeling, a computational model of web application can be built. In this model, web pages can be considered as states of finite state model and links can be considered as state transitions with input condition provided at each state. And then with the help of WP method and UIO sequence method, test cases can be generated from the minimal, complete, and connected FSM [2]. In this, the mutants or wrong input conditions can be inserted to check validity and security of the web application.

REFRENCES

[1] Roger S. Pressman, Sixth Edition, International Edition 2005, Mc Graw Hill, "Software Engineering: A practitioner's approach", ISBN: 007-124083-7, pp. 595-600.
[2] Aditya P. Mathur, Purdue University, "Foundation of Software Testing", ISBN-10 8131759083, Edition 2011.
[3] Anneliese A. Andrews, Jeff Offutt, Roger T. Alexander" Testing Web Applications by modeling with FSMs", In IEEE proceedings of the Ninth Annual Conference on Computer Assurance on june, 1994.
[4] Bo Song, Huaikou Miao, Zhongyu Chen, "Modeling Database Interactions in Web applications and Generating Test Cases", In IEEE World Congress on Software Engineering, 2007.
[5] Bo Song, Shengwen Gong, Shengbo Chen, "Model Composition and Generating Tests for Web Applications", In Seventh IEEE International Conference on Computational Intelligence and Security, 2011.
[6] Hierons R.M. & Merayo M.G., "Mutation Testing from Probabilistic Finite State Machine", In IEEE Conference on Computer Assurance, 2007.
[7] Ana Cavalli, Tien-Dung Cao, Wissam Mallouli, Eliane Martins4,Andrey Sadovykh, Sebastien Salva, Fatiha Za di, "WebMov: A dedicated framework for the modelling and testing of Web Services Composition", In International Conference on Web Services, 2010.
[8] A. S. Kalaji, R. M. Hierons & S. Swift, "Generating Feasible Transition Paths for Testing from an Extended Finite State Machine", In IEEE International Conference on Software and Information Technology, 2009.
[9] A. S. Kalaji, R. M. Hierons & S. Swift, "Mutation Testing of Probabilistic Finite State Modelling", In international Conference on Information and Software Technology, 2011.
[10] Xi Wang, Liang Guo, Huaikou Miao, "An Approach to Transforming UML Model to FSM Model for Automatic Testing", In International Conference on Computer Science and Software Engineering, 2008.
[11] Liping Li, QIAN Zhongsheng, Tao He, "Test Purpose-Based Test Generation for Web Applications", In IEEE International Conference on Web Services, 2009.