



Testing of Web Applications using UIO Sequence Method

Kulvinder Singh and Semmi

UIET, Kurukshetra University Kurukshetra ,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 challenging. This paper addresses the use of unique input/output sequence method (UIO) based on finite state machine (FSM). A UIO sequence is a sequence of input and output pairs that distinguishes a state of the FSM from the remaining states. According to the UIO sequence method, it details the process of generating test cases by finding unique sequence for each state of the FSM representation of web site. It also details test optimization by reducing the set of test subsequences.

Keywords: Web Application, UIO Input Output Sequence, Finite State Machine, Test Cases, Test Optimization

I. 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 is concerned with the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to successful development, deployment and maintenance of high quality web based systems and applications. 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 [1].

A. 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. Finally, the focus shifts to tests that exercise technological capabilities [1].

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.

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.

Navigation testing

First phase of navigation testing actually begins during interface testing where we test all navigation and interaction mechanisms. And we test all navigation semantic units (NSU) to ensure that each NSU can be achieved by appropriate user category.

Component Level Testing:

This type of testing focuses on a set of tests that attempt to uncover errors in WebApp functions. Each software function within the WebApp considered as a software module or component that can be tested using conventional testing techniques like Black box and White box.

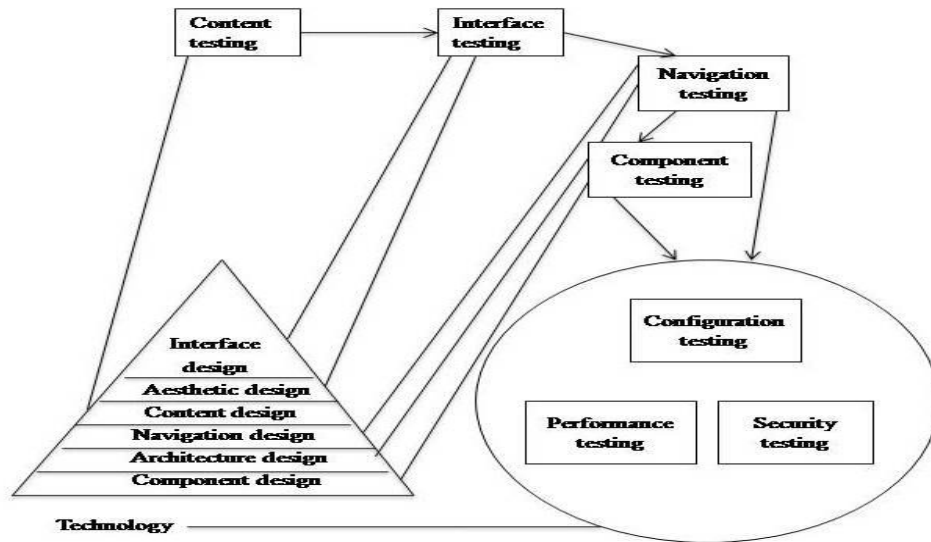


Fig - 1. Testing process

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.

Security testing:

It incorporates a series of tests designed to uncover weaknesses in the WebApp and its environment that can be exploited by those with malicious intent. To protect against these vulnerabilities, one or more of the security elements can be implemented for e.g. Firewall, Authentication, Encryption, and Authorization.

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].

II. Finite State Machine

A. Introduction

Finite state machines are primitive but useful computational model for both hardware and certain types of software. Design criteria of finite state machines can also easily be expressed in terms of states and state transitions. The formal model of a communicating finite state machine plays an important role in three different areas of protocol design: formal validation, protocol synthesis, and conformance testing. A finite state machine, abbreviated as FSM, is an abstract representation of behavior exhibited by some systems. An FSM is derived from application requirements. For example, a network protocol could be modeled using an FSM. Not all aspects of an application's requirements are specified by an FSM [2].

B. FSM: Definition [2]

An FSM is a 6-tuple machine $(X, Y, Q, q_0, \delta, O)$, where :

X is a finite set of input symbols also known as the input alphabet.

Y is a finite set of output symbols also known as the output alphabet.

Q is a finite set of states.

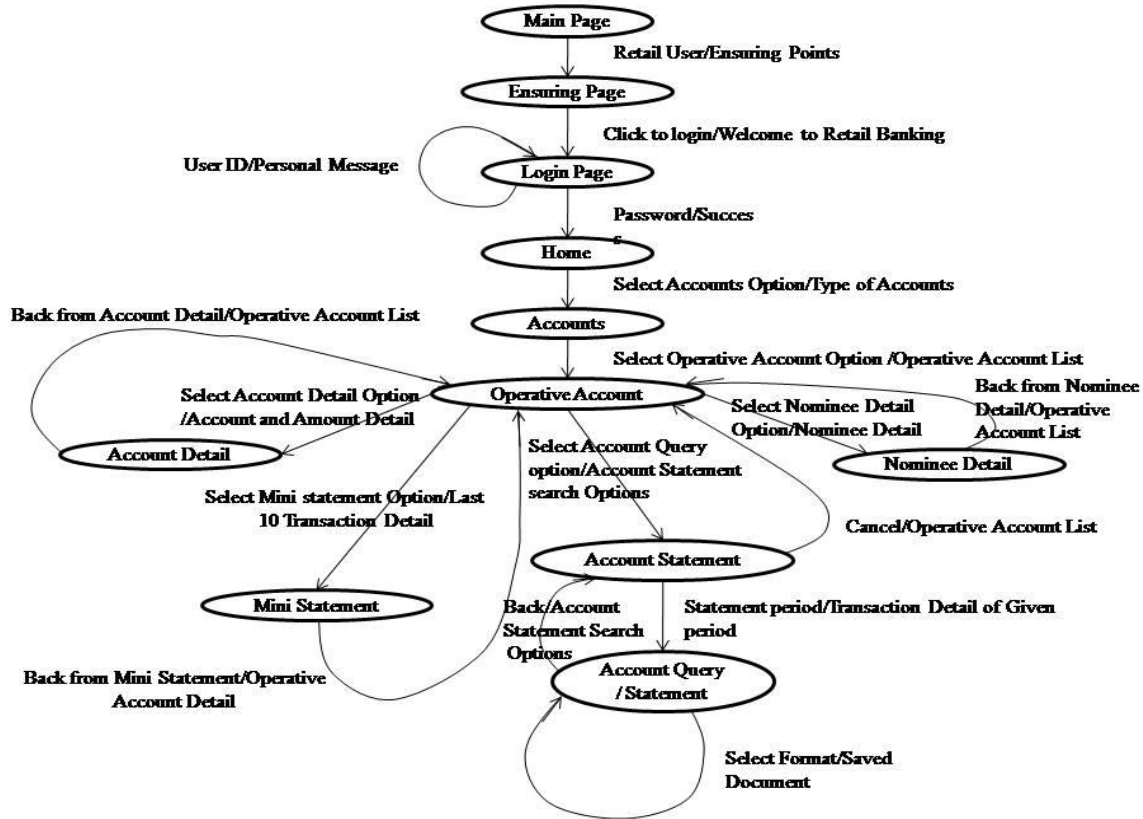
$q_0 \in Q$ is the initial state.

$\delta : Q \times X \rightarrow Q$ is a next state or state transition function.

$O : Q \times X \rightarrow Y$ is an output function.

C. Modeling Web Application as FSM

We create a finite state diagram of subsystem of Banking Web Application having various states with their appropriate input and outputs shown below. From this FSM diagram of banking subsystem we can generate UIO sequences for each state by applying UIO sequence generation algorithm. As we create FSM diagram of a module of banking Web Application, we can create FSM diagram of other modules of banking Web Application and likewise we can generate UIO sequences for those modules further. This FSM diagram consists of various states of online banking with their appropriate input and output labels. Each state after receiving input move to a next state which can be another or same state and produce a output [2].



FSM diagram of banking web application subsystem

III. UIO Sequence Method

UIO Sequences: A UIO sequence is a sequence of input and output pairs that distinguishes a state of an FSM from the remaining states. Consider FSM $M = (X, Y, Q, q_0, \delta, O)$. A UIO sequence of length k for some state $s \in Q$, is denoted as $UIO(s)$ and looks like the following sequence[4]:

$$UIO(s) = i_1/o_1 . i_2/o_2 . . . i_{(k-1)}/o_{(k-1)} . i_k/o_k .$$

In the sequence given above, each pair i_1/o_1 consists of an input symbol i_1 that belongs to the input alphabet X and an output symbol o_1 that belongs to the output alphabet Y . The dot symbol $(.)$ denotes string concatenation. We refer to the input portions of a $UIO(s)$ as $in(UIO(s))$ and to its output portion as $out(UIO(s))$.

$$in(UIO(s)) = i_1 . i_2 . . . i_{(k-1)} . i_k \text{ and}$$

$$out(UIO(s)) = o_1 . o_2 . . . o_{(k-1)} . o_k$$

A formal definition of UIO Sequence is as follows:

Given an FSM $M = (X, Y, Q, q_0, \delta, O)$, the UIO sequence for state $s \in Q$, denoted as $UIO(s)$, is a sequence of one or more edge labels such that the following condition holds[3].

$$\delta(s, in(UIO(s))) \neq \delta(t, in(UIO(s))), \text{ for all } t \in Q, t \neq s.$$

This condition shows that output state of both transition function must be different for both $UIO(s)$.

A. Acronyms for various states, input and outputs labels:

We assume acronyms for various states and input, outputs labels shown above in FSM diagram for easier implementation. We mentioned these assumptions in tabular form below [3].

States	Acronym
Main Page	q1
Ensuring Page	q2
Login Page	q3
Home	q4
Accounts	q5
Operative Accounts	q6
Account Detail	q7
Mini Statement	q8
Account Statement	q9
Account Query/Statement	q10
Nominee Detail	q11

Edge	Input/output
q1, q2	RU/EP
q2, q3	CTL/WRB
q3, q3	UI/PM
q3, q4	PWD/S
q4, q5	SAO/TOA
q5, q6	SOAO/OAL
q6, q7	SADO/AAD
q7, q6	BFAD/OAL
q6, q8	SMO/TD
q8, q6	BFMS/OAL
q6, q9	SAQO/ASSO
q9, q6	C/OAL
q6, q11	SNDO/ND
q11, q6	BFND/OAL
q9, q10	SP/TDGP
q10, q9	B/ASSO
q10, q10	SF/SD

B. Generation of UIO Sequences:

The UIO Sequences are essential to the UIO-sequence method for generating tests from a given FSM. An algorithm for generating Uio sequences for all states of a given FSM is as follows [4].

Input: (a) An FSM $M = (X, Y, Q, q_0, \delta, O)$, where $|Q| = n$.

(b) State $s \in Q$.

Output: $UIO[s]$ contains a UIO sequences for state s ; $UIO(s)$ is empty if no such sequence exists.

Procedure: gen-uio(s)

Step 1 For each distinct edge label el in the FSM, compute $Set(el)$.

Step 2 Compute the $Set\ Oedges(s)$ of all outgoing edges for state s . Let $NE = |Oedges|$

Step 3 For $1 \leq i \leq NE$ and each edge $e_i \in Outedges$, compute $Oled[i]$, $Opattern[i]$, and $Oend[i]$ as follows:

3.1 $Oled[i] = Set(label(e_i)) - \{e_i\}$

3.2 $Opattern[i] = label(i)$

3.3 $Oend[i] = tail(i)$

Step 4 Apply algorithm gen-1-uio(s) to determine if $UIO[s]$ consists of only one label.

Step 5 If simple UIO sequence found then return UIO[s] and terminate this algorithm, otherwise proceed to the next step.

Step 6 Apply the gen-long-UIO(s) procedure to generate longer UIO[s].

Step 7 If longer UIO sequence found then return UIO[s] and terminate this algorithm, else return with an empty UIO[s].

UIO Sequences: UIO sequences for each state of FSM diagram of subsystem of banking website is given below.

States	UIO[s]	q6	SADO/AAD
q1	RU/EP	q7	BFAD/OAL
q2	CTL/WRB	q8	BFMS/OAL
q3	PWD/S	q9	C/OAL
q4	SAO/TOA	q10	B/ASSO
q5	SOAO/OAL	q11	BFND/OAL

C. Test Generation:

1. Find the UIO for each state in M.
2. Find the shortest path from the initial state to each of the remaining states. As mentioned earlier, the shortest path from the initial state q_1 to any other state $q_i \in Q$ is denoted by $P_i(q_1)$.
3. Construct the edge tour for each edge in M. Let $TE(e)$ denote a subsequence that generates a tour for edge e . $TE(e)$ is constructed as follows[4]:
 $TE(e) = P_{head(e)}(1).label(e).UIO(tail(e)).$

Test Count	Edge(s)	TE(e)
1	q_1, q_2	RU/EP.CTL/WRB
2	q_2, q_3	RU/EP.CTL/WRB.PWD/S
3	q_3, q_4	RU/EP.CTL/WRB.PWD/S.SAO/TOA
4	q_4, q_5	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL
5	q_5, q_6	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SADO/AAD
6	q_6, q_7	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.BFAD/OAL
7	q_7, q_8	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SMO/TD.BFMS/OAL
8	q_8, q_9	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.C/OAL
9	q_9, q_{10}	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.B/ASSO
10	q_{10}, q_{11}	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.B/ASSO.C/OAL
11	q_{11}, q_1	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO
12	q_1, q_2	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO
13	q_2, q_3	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO
14	q_3, q_4	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO
15	q_4, q_5	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO
16	q_5, q_6	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO
17	q_6, q_7	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO

D. Test Optimization:

The set of test subsequences $TE(e)$ can often be reduced by doing a simple optimization. For example, if $TE(e_1)$ is a subsequence of test $TE(e_2)$, then $TE(e_1)$ is redundant. This is because the edges toured by $TE(e_1)$ are also toured by $TE(e_2)$, and in the same order. Identification and elimination of subsequences that are fully contained in another subsequences generally leads to a reduction in the size of the test suite. In addition, if two tests are identical then one of them can be removed. Hence, Test Optimization of the test cases generated above are given below. We examine each test and check if it is contained in any of the remaining tests. We find that test 1, test 2, test 4, test 5, and test 6 are fully contained in test 11, and test 8 is contained in test 12, and test 9 is contained in test 13 and test 10 is contained in test 14, and test 15 is contained in test 16 [4].

Test Count	Edge(s)	TE(e)
1	q_3, q_3	RU/EP.CTL/WRB.UI/PM.PWD/S
2	q_6, q_7	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.BFAD/OAL
3	q_7, q_6	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SADO/AAD.BFAD/OAL.SADO/AAD
4	q_8, q_6	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SMO/TD.BFMS/OAL.SADO/AAD
5	q_9, q_6	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.C/OAL.SADO/AAD
6	q_{11}, q_6	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SNDO/ND.BFND/OAL.SADO/AAD
7	q_{10}, q_9	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.B/ASSO.C/OAL
8	q_{10}, q_{10}	RU/EP.CTL/WRB.PWD/S.SAO/TOA.SAO/OAL.SAQO/ASSO.SP/TDGP.SF/SD.B/ASSO

IV. Conclusion

In this paper we presented the test case generation by using UIO sequence method based on FSM. A UIO sequence is a sequence of input and output pairs that distinguishes a state of the FSM from the remaining states. We made a FSM diagram of a module of web site and generated unique sequence of input and output pairs for each state. After that we found shortest path from initial state to remaining states and then by using this shortest path we constructed the edge tour for each edge e in the FSM. This edge tour for each edge represents various test cases for the subsystem considered. We also optimized this test suite of test cases by identifying and eliminating redundant test cases with in the test suite. This optimized test suite can help in finding as much errors as possible in the interface of a web application.

References

- [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] Hopcroft, John E. and Jeffrey D. Ullman. 1979. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA.
- [3] Kulvinder Singh et. al. "Testing Web Based Applications Using Finite State Machine Employing Genetic algorithm", International Journal of Engineering Science and Technology, Vol. 2(12), 2010, 6931-6941
- [4] Aditya P. Mathur, Purdue University, "Foundation of Software Testing", ISBN-10 8131759083, Edition 2011.
- [5] 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.