Journal of Software Engineering and Applications, 2012, 5, 7-13
http://dx.doi.org/10.4236/jsea.2012.51002 Published Online January 2012 (http://www.SciRP.org/journal/jsea)

Web Testing Generation: A Stream X-Machine Based Approach

Zhongsheng Qian[1,2]

[1]School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, China; [2]Strong Digital Technology Co., Ltd., Nanchang, China.
Email: changesme@163.com

Received November 26th, 2011; revised December 31st, 2011; accepted January 1st, 2012

ABSTRACT

To ensure the quality of Web applications, Web testing is one of the effective methods. The testing is a process of revealing errors that is used to give confidence that the implementation of a Web application meets its original specification. This work proposes a Web testing framework based on Stream X-Machines (SXMs), which provides a way to derive test cases for a Web application. It starts from constructing the SXM model, from which a test translator is employed to extract the test paths and then translates them into an XML-style test specification, which is the input of test engine. The test engine generates test cases and then executes them, and finally produces test report. This testing method is a significant contribution to informed research.

Keywords: Web Application; SXM (Stream X-Machine); FSM (Finite State Machine); Test Case; Testing Framework

1. Introduction

A wide range of important activities are supported by Web applications. Given the importance of such activities, bad Web applications can have far-ranging consequences on businesses, economies, scientific progress, health, and so on. Web testing is an effective technique to ensure the quality of Web applications. Traditional testing approaches are no longer adequate for Web applications. Web applications typically undergo maintenance at a faster rate than other software systems and this maintenance often consists of small incremental changes [1]. To accommodate such changes, Web testing approaches must be automatable and test sets must be adaptable. However, Web applications raise important and challenging test issues that cannot be solved directly by existing test techniques for conventional programs [2,3].

The problem of test effectiveness is best addressed if the test set can be guaranteed to find all faults of the implementation. One approach is to consider two algebraic objects (the specification and the implementation), each of them characterized by an input/output behavior, and to prove that, if the behaviors of these objects coincide for any input in the test set, they will coincide for any input in the domain. Thus, the specification and the implementation will be guaranteed to have identical behavior provided that they behave identically when supplied with the inputs in the test set. A considerable amount of work by this approach has been employed in the area of test generation for software modelled by Finite State Machines (FSMs). Here, the assumption is that the control aspects of the software can be somehow separated from the system data and can be modelled as an FSM. However, for non-trivial systems, it is usually impossible to describe the system control independent of its data processing. Furthermore, it may be very difficult, or even impossible, to derive the control structure from a system specification, unless appropriate specification languages are used. For example, an FSM can be derived from a VDM specification only under certain conditions and, even in this case, the resulting machine may be of an unmanageable size. Therefore, a more complex specification model that integrates these two aspects is needed. Such a model is the X-machine, a blend of FSMs, data structures and processing functions.

Essencially, an X-machine is like an FSM, but with one important difference: the labels of the transitions are (partial) functions instead of abstract symbols. Of particular practical importance are those X-machines, called Stream X-Machines (SXMs), where these functions process input and output symbols while changing the value of an internal memory or data set *M*. *M* is often an array consisting of fields such as registers, stacks, database filestores, etc., so it is possible to model very general systems in a transparent way. SXMs are, however, best suited for specifying interactive systems such as Web applicatiions.

We discuss, in this paper, the testing problems for Web applications based on SXMs. The SXM-based testing method does not rely on the finiteness of the memory set, for there is no need to construct the equivalent FSM and furthermore, it avoids the state explosion problem.

2. Stream X-Machines

SXMs are special instances of the X-machines introduced in 1974 by S. Eilenberg [4]. They employ a diagrammatic method of modelling control flow by extending the expressive power of FSMs. Compared to an FSM, instead of using abstract symbols, the labels of the transitions in a SXM are *relations* (often *partial functions*) that operate on a basic data set X. The set of these relations, Φ, is called the *type* of the SXM and represents the elementary operations that the SXM is capable of performing. In SXMs, all data are triples consisting of a stream of input symbols, a stream of output symbols and an internal memory value.

The basic idea is that the SXM has some internal memory, M, and the stream of inputs determine, depending on the current state of control and the current state of the memory, the next control state, the next memory state and the output value. SXMs [5,6] are a computational model capable of representing both the data and the control of a system. The computation of the SXM starts in a given initial state (control state) and a given state of the system's underlying data set X (the data state).

In **Figure 1**, for example, there are a number of paths that can be traced out from the initial state q_1 and each edge is labelled by a relation: φ_1, φ_2, φ_3, etc. Sequences of relations are thus derived from each path in the state space and these may be composed to produce a relation that may be defined on the data state. This is then applied to the value x ($\in X$) providing that the composed relation is defined on x. This then gives a new value, x (if there is more than one such x, then one will be picked in a non-deterministic way) for the data state and a new control state.

Definition 2.1 A Stream X-Machine (SXM) is a nonuple $Z = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0)$, Where:

- Σ and Γ are finite sets called the *input alphabet* and *output alphabet* respectively.
- Q is the finite set of *states*.
- *M is a (possibly) infinite set called memory.*
- Φ is the *type*, a finite set of distinct non-empty *processing relations* of the form $\varphi : M \times \Sigma \leftrightarrow \Gamma \times M$; Φ is often a set of (partial) functions.
- F is the (partial) *next state function* (or *state transition function*), F: $Q \times \Phi \rightarrow 2^Q$; as for finite automata, F is usually described by a *State Transition Diagram* (*STD*).
- I and T are the sets of initial and terminal states respectively, $I \subseteq Q, T \subseteq Q$.

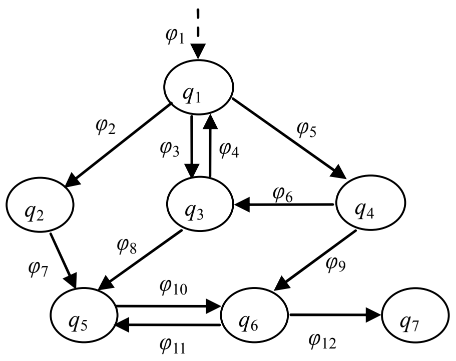

Figure 1. The state transition diagram of a SXM.

- m_0 is the initial memory value, $m_0 \in M$.

Thus, SXMs are X-machines, of which each processing relation will read an input symbol, then discard it and produce one or more output symbols while (possibly) changing the value of the memory.

Definition 2.2 Given a *SXM* $Z = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m_0)$, the finite automaton $FA_Z = (\Phi, Q, F, I, T)$ over the alphabet Φ is called the *associated FA* of Z.

It is sometimes helpful to regard an X-machine as a finite automaton with the arcs labelled by relations from the type Φ.

Definition 2.3 A SXM Z is called *completely defined* if $\forall q \in Q$, $m \in M$, $\sigma \in \Sigma$, $\exists \varphi \in \Phi$ such that $(m, \sigma) \in dom(\varphi)$ and $(q, \varphi) \in dom(F)$ hold.

A completely defined SXM is one, in which there is at least one possible transition for any triplet $q \in Q$, $m \in M$, $\sigma \in \Sigma$. A SXM that is not complete is assumed to ignore an input, which does not cause any transition, by remaining in the same state with an unchanged memory value.

In contrast to FSMs, SXMs are capable of modelling non-trivial data structures by employing a memory attached to the state machine. Moreover, transitions between states are not labelled with simple input symbols but with processing functions. Processing functions receive input symbols and read memory values, and produce output symbols while modifying memory values. The benefit of adding the memory construct is that the state explosion is avoided and the number of states is reduced to those states which are considered critical for the correct modelling of the system's abstract control structure. A *divide-and-conquer* approach to designing allows the model to hide some of the complexity in the transition functions, which are later exposed as simpler SXMs at the next level.

Apart from being formal and proven to possess the computational power of Turing machines [5], SXMs offer a highly effective testing method for verifying the conformance of a system's implementation against a specification. SXM models can be represented in XMDL (X-Machine Definition Language), a special-purpose markup language introduced by E. Kapeti and P. Kefalas [7]. XMDL has served as a common language for the development of numerous tools supporting SXMs [8].

3. Testing Generation Based-On Stream X-Machines

A SXM-based testing method [5,9] that extends the W-method [10], enables us to derive a complete finite set of test cases that is proven to find all faults in the implementation. The outcome of the test generation algorithm is a finite set of input and expected output sequences. The inputs and expected outputs need to be mapped to concrete executable test cases that can be processed by a testing engine, in order to interact with the Web component under test and provide the results. The main point about SXM testing, the type of test generation technique discussed here, however, is to establish, not only efficient test sets from a specification but also to relate the test process to a hierarchical decomposition of the implementation in order to make the management of the test process more convenient.

In this work, it is assumed that a tester applies a test sequence as a whole and subsequently checks an output sequence produced by an implementation, rather than applying an input sequence symbol-by-symbol and observing output symbols as they come out.

Definition 3.1 In a SXM *Z*, if q, $q' \in Q$, $\varphi \in \Phi$, and $q' \in F(q,\varphi)$, we say that φ is an *arc* from q to q' of *Z* and denote as $\varphi : q \to q'$.

In a SXM *Z*, for $\forall q \in Q$, if any other state is reachable directly from *q*, then *q* is called a *to-all state*; if any other state can not be reached from *q*, then *q* is called a *to-none state*; if *q* is reachable directly from any other state, then *q* is called an *all-to state*; if *q* can not be reached from any other state, then *q* is called a *none-to state*; if *q* is a *to-none state*, also a *none-to state*, then *q* is called an *isolative* (*or starved*) *state*. An *isolative state* is not allowed in a Web testing model, for users of a Web application can never stay in an *isolative state*.

Definition 3.2 In a SXM *Z*, if q, $q' \in Q$ are such that $\exists q_1,\cdots,q_{n+1} \in Q$ with $q_1 = q$ and $q_{n+1} = q'$ so that φ_1: $q_1 \to q_2$, φ_2: $q_2 \to q_3$, $\cdots$, φ_n: $q_n \to q_{n+1}$, we say that we have a (test) path $p = \varphi_1 \cdots \varphi_n$ from q to q' of *Z* and denote as *p*: $q \to q'$. Each path $p = \varphi_1 \cdots \varphi_n$ gives rise to a (partial) function (or the path function) [*p*]: $M \times \Sigma^* \leftrightarrow \Gamma^* \times M$ defined by

$[p]$ $(m,s) = (g,m')$ if $\exists n \ge 0$, $\sigma_1,\cdots,\sigma_n \in \Sigma$, γ_1, $\cdots,\gamma_n \in \Gamma$, $m_1,\cdots,m_{n+1} \in M$ with $m_1 = m$, $m_{n+1} = m'$, $s = \sigma_1,\cdots,\sigma_n$ and $g = \gamma_1,\cdots,\gamma_n$ so that $\varphi_i(m_i,\sigma_i) = (\gamma_i, m_{i+1})$, $\forall 1 \le i \le n$. The functioon corresponding to the empty path τ is defined by $[\tau]$ $(m,\tau) = (\tau, m)$, $m \in M$.

The sequence of transitions (path) caused by the stream of input symbols is called a computation. The computation halts when all input symbols are consumed. The result of a computation is the sequence of outputs produced by this path.

Example 1. In **Figure 1**, there are four test paths from q_1 to q_6. These test paths are $\varphi_1\varphi_2\varphi_7\varphi_{10}$, $\varphi_1\varphi_3\varphi_8\varphi_{10}$, $\varphi_1\varphi_5\varphi_9$ and $\varphi_1\varphi_5\varphi_6\varphi_8\varphi_{10}$. In each of these test paths, it is usual to generate more than one test cases from q_1 to q_6.

Definition 3.3 Two deterministic SXMs (DSXMs [4]) *Z* and *Z'* are called *weak testing compatible* if they have identical input alphabets, output alphabets, memory sets and initial memory values.

Definition 3.4 Two weak testing compatible DSXMs are called *testing compatible* if they have identical types.

The basic idea of the method is to translate test sets of the associated FA into test sets of the DSXM specification. SXMs are a subclass of X-machines [4] that extend FSMs and are supported by a test generation method that is guaranteed to reveal all faults in an implementation under test, given that certain realistic conditions hold. This works if the DSXM specification satisfies two conditions: *input-completeness* (*i.e.*, all processing functions can be exercised from any memory value using appropriate inputs) and *output-distinguishability* (*i.e.*, any two different processing functions will produce different outputs if applied on the same memory/input pair).

Definition 3.5 Φ is called *input-complete* if $\forall \varphi \in \Phi$, $m \in M$, $\exists \sigma \in \Sigma$ such that $(m,\sigma) \in dom(\varphi)$.

This condition ensures that any processing function can be exercised from any memory value using appropriate input symbols.

Definition 3.6 A SXM is *output-distinguishable* if $\forall \varphi_1, \varphi_2 \in \Phi$, $\varphi_1 \ne \varphi_2$, if $\varphi_1(m,\sigma) = (\gamma_1, m_1)$ and $\varphi_2(m,\sigma) = (\gamma_2, m_2)$ then $\gamma_1 \ne \gamma_2$.

The *output-distinguishability* condition can also be described in the following.

Definition 3.7 Φ is called *output-distinguishable* if $\forall \varphi_1, \varphi_2 \in \Phi$, $((\exists m \in M, \sigma \in \Sigma$ with $\pi_1(\varphi_1(m,\sigma)) = \pi_1(\varphi_2(m,\sigma))) \Rightarrow \varphi_1 = \varphi_2)$.

This says that we must be able to distinguish between any two different processing functions by examining outputs. If we cannot then we will not always be able to tell them apart.

These two conditions (output-distinguishability and input-completeness) are generally known as *design for test conditions* [9]. The output-distinguishability condition ensures that any processing function can be identified from the machine computation by examining the outputs produced. The input-completeness condition ensures that all sequences of processing functions in the associated FA can be exercised using appropriate inputs, so they can be tested against the implementation. Without the described conditions, it would be extremely difficult to test a Web application properly. For instance, if a method generates a particular sequence of functions to execute and there is no corresponding sequence of inputs to attempt this sequence, such a sequence cannot be executed. As a result, some faults of the implementation may go undetected.

There are often many test cases in testing a Web application, for representing a test set in a reduced way, we introduce the following several notations:

- $[tc_1, tc_2]$, which says that either tc_1 or tc_2 is selected to execute each time.
- (tc_1, tc_2), which says that both tc_1 and tc_2 are executed in any order.
- $\langle tc_1, tc_2 \rangle$, which says that both tc_1 and tc_2 are executed and tc_1 is executed before tc_2.

For example, a test set may be written as $\langle tc_1, tc_2, [(tc_3, tc_4), [tc_5, tc_6]], tc_7 \rangle$, which is the reduced representation of these four test sets $\langle tc_1, tc_2, tc_3, tc_4, tc_7 \rangle$, $\langle tc_1, tc_2, tc_4, tc_3, tc_7 \rangle$, $\langle tc_1, tc_2, tc_5, tc_7 \rangle$ and $\langle tc_1, tc_2, tc_6, tc_7 \rangle$.

The above-mentioned testing method makes it possible to find a test input for every memory value. Instead, it could be possible only to use a subset of inputs and memory values for testing which could make attempting elements of Φ easier in practice. For a relation φ, inputs used for testing are denoted $U\varphi$; memory values traversed during testing are assumed to be contained in $V \subseteq M$. Usage of a subset of inputs is only possible if

- Memory of an X-machine under test will stay within V.
- It is possible to attempt every relation $\varphi \in \Phi$ using a subset of inputs.
- Outputs from these relations make it possible to distinguish between them.

The above three items are formalized using the idea of a *testing context* (V,U) [11]. The purpose of $U\varphi$ is to contain inputs to attempt φ ($\varphi \in U$).

Definition 3.8 Let (V, U) be such that $V \subseteq M$ and $U = \{U\varphi \subseteq \Sigma \mid \varphi \in \Phi\}$. The set Φ is called *closed* w.r.t (V, U) if the following two conditions are satisfied,

- $m_0 \in V$.
- $\forall \varphi \in \Phi$, $m \in V$, $\sigma \in U\varphi$, if there is $\gamma \in \Gamma$, $m' \in M$ such that $\varphi(m, \sigma) = (\gamma, m')$, then $m' \in V$.

Definition 3.9 Let Φ be closed w.r.t. some (V, U) where $V \subseteq M$ and $U = \{U\varphi \subseteq \Sigma \mid \varphi \in \Phi\}$. Then Φ is *input-complete* w.r.t (V, U) if $\forall \varphi \in \Phi$, $m \in V$, $\exists \sigma \in U\varphi$ such that $(m, \sigma) \in dom(\varphi)$.

Definition 3.10 Let Φ be closed w.r.t some (V, U) where $V \subseteq M$ and $U = \{U\varphi \subseteq \Sigma \mid \varphi \in \Phi\}$. Then Φ is *output-distinguishable* w.r.t (V, U) if $\forall \varphi_1, \varphi_2 \in \Phi$, $m \in V$, $\sigma(U\varphi_1 \cap dom(\varphi_2)) \cup (U\varphi_2 \cap dom(\varphi_1))$, $\exists \gamma \in \Gamma$, $m_1', m_2' \in V$. if $\varphi_1(m, \sigma) = (\gamma, m_1')$ and $\varphi_2(m, \sigma) = (\gamma, m_2')$, then $\varphi_1 = \varphi_2$.

For $V = M$ and a DSXM, the above two definitions reduce to input-completeness and output-distinguishability; the set Φ of any SXM is closed w.r.t (M, Σ) by definition of a SXM. For these reasons, Definitions 3.9 and 3.10 are more general than Definitions 3.5 and 3.6 (or 3.7).

Definition 3.11 Let $V \subseteq M$ be a non-empty subset of memory values and $n \geq 1$. The SXM is called *n-complete* w.r.t. V if there exist $V_0 \subseteq V_1 \cdots \subseteq V_{n-1} \subseteq M$, $V_0 = V$, such that for any $\varphi \in \Phi$,

- if $m \in V_{n-1}$ then there exists $\sigma \in \Sigma$ such that $(m, \sigma) \in dom(\varphi)$.
- if $\varphi(m, \sigma) = (\gamma, m')$ with $m \in V_{i-1}$ and $(m, \sigma) \in dom(\varphi)$ then $m' \in V_i$, $1 \leq i \leq n-1$.

The above condition guarantees that any path of length at most n of the associated automaton can be exercised by an input test sequence from the initial state and an initial memory value in V.

4. A Feasible SXM-Based Web Testing Framework

It is widely accepted that models constitute important mechanism in the process of a Web application development and help us understand the system by omitting some details.

To test a Web application, test cases must be generated. Our test paths produced can be easily employed to construct test cases. A test case defined by our Web testing framework (cf. **Figure 2**) is one test path with user input values. So, a test path may be used to construct multiple test cases if only the tester provides different user input values. Our test specification is an extended version of the one described in [12], where the specification is depicted using XML. The test specification is based on request specification, response specification, and predicate definition. Request specification specifies a pattern of HTTP requests, while response specification specifies the assertions on the HTTP response generated from the HTTP request in the same test step, and finally, the predicate definition specifies the assertions on the results of testing.

The Web testing framework extends the testing approach proposed by X. Jia, *et al.* [12]. At the beginning, the SXM is constructed through the commonly-used requiring and analyzing methods of Web applications. The test translator of the testing framework extracts component relations from the SXM. It then translates the test paths into test script frame (of test specification). This frame defines the test set, containing test cases. Every test case is a sequence of test steps. Each test step is for one component to be verified. Every test step defines HTTP requests, expected response and predicates with condition definitions (such as <not>, <and>, <or>, <match>, and <forall>). The test script frame does not contain values for input variables, since they will be added later. Some inputs have to be created by hand, including user ids and passwords. Other inputs are either automatically extracted

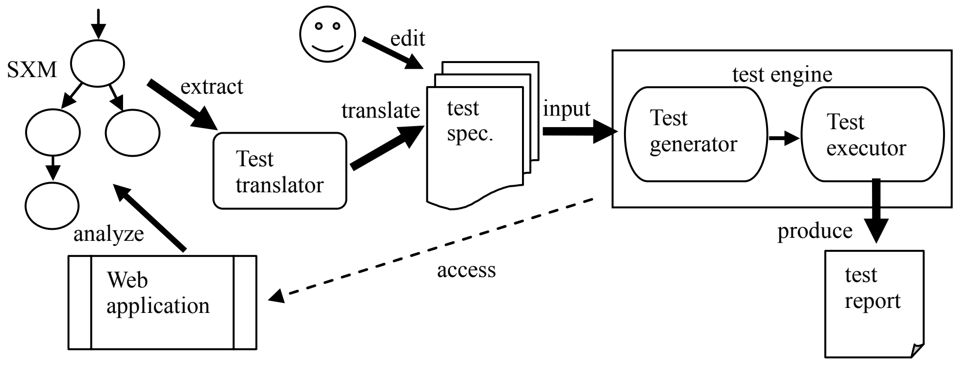

Figure 2. A SXM-based web testing framework.

from the HTML files or randomly generated. Moreover, the test script frame does not contain hard-coded expected output, that is to say, the Web testing framework defines frames with empty response, since they will be filled manually by the tester.

After editing the necessary information of the test script frame, the tester gets a formal XML-style specification, which is the input to test engine. The specification contains templates of test cases. Test engine includes a test case generation engine (test generator) that is able to determine the test paths from the test specification, and to generate test cases from it, provided that a test criterion is specified. Generated test cases are sequences of links which, once executed, grant the coverage of the selected criterion. Then the test engine executes the test cases and validates its result against the test oracles specified as expected result. That is to say, test engine's test executor can now provide the link sequences of each test case to the Web server, attaching proper inputs to each form. After the execution, test engine produces a test report summarizing the results of all test cases. For such evaluation, the tester opens the output pages on a Web browser and checks whether the output is correct for each given input. By the way, the test engine behaves as a Web client accessing the Web application.

5. Related Work

Many Web testing challenges are discussed in [2,13], and a number of Web testing techniques for Web applications have been already proposed [14-21], each of which has different origins and pursues different test goals for dealing with the unique characteristics of Web applications.

Andrews, *et al.* [15] illustrated an approach to modeling and testing Web applications based on FSMs after analyzing eight kinds of connections among Web pages and software components of Web applications. They partitioned a Web application into several functional clusters and logical pages, and tried to use hierarchical constrained FSMs to represent the logical pages and their navigations. However, the interactions and composition of components are not considered further.

Elbaum, *et al.* [17] proposed a method to use what they called *user session data* to generate test cases for Web applications. Instead of looking at the data kept in J2EE servlet session, their *user session data* is the input data collected and remembered from previous user sessions. The *user session data* is captured from HTML forms and includes name-value pairs. Our approach is flexible, and the user input data can be produced by various methods presented by existing research work.

Ricca and Tonella [18] suggested a UML model of Web applications and proposed that all paths that satisfy selected criteria should be tested. They also presented an analysis model and corresponding testing strategy. Their strategy is mainly based on static Web page analysis and some preliminary dynamic analysis. Liu, *et al.* [19] extended traditional data flow testing techniques to support Web application testing. A test model, WATM, which consists of an object model and a structure model, is presented to capture the data flow information of Web applications. These studies [18,19] consider only the underlying structure and semantics of Web applications towards a white-box testing approach. They focus on the internal structural aspect and involve in the details of a Web application.

Object driven performance testing was proposed by Subraya and Subrahmanya [20]. They illustrated a new testing process that employs the concept of decomposing the behavior of a Web application into testable components.

A survey of Web application testing was given by Lucca and Fasolino [3]. They presented the main differences between Web applications and traditional ones, how these differences impact the testing of the former, and some relevant contributions in the field of Web application testing developed in recent years.

SXM-based testing has been developed in various directions. Moreover, numerous case studies have been carried out to establish if the approach is actually practical. Thus, a number of systems have been specified as SXMs and the tests produced by the approach used to validate the implementations.

Kourtesis, *et al*. [22] presented an approach towards effective service discovery and selection. They employed SXMs as a powerful modelling formalism for constructing the behavioural specification of a Web service, for performing verification through the generation of exhaustive test cases, and for performing validation through animation or model checking during service selection.

Ipate and Holcombe [23] provided a new variant of the SXM-based testing method that no longer depends on the size of a *controllable* model of the IUT. In data processing-oriented applications, the new method can drastically reduce the size of the test suite produced at the expense of a (possibly) more complex generation process.

Merayo, *et al*. [24] presented a formal testing framework for systems where timeouts are critical. The model introduced for specifying the systems is a suitable extension of the classical concept of SXM. They introduced a notion of test that can delay the execution of the implementation and also proposed an algorithm to derive sound and complete test sets.

Bogdanov, *et al*. [25] described the X-machine testing method and its use for testing of different types of systems, both in terms of theory and practical outcomes. They surveyed the extensions of the X-machine testing method for testing of functions together with testing of a transition diagram, equivalence testing of a non-deterministic implementation against a non-deterministic specification, conformance testing of a deterministic implementation against a non-deterministic specification and equivalence testing of a system of concurrently executing and communicating X-machines, against a specification.

Ipate and Gheorghe [26] presented the complete non-deterministic SXM (NSXM) testing method to generalize the NSXM integration testing method. It no longer requires implementations of the processing relations to be proved correct before integration testing can take place. Instead, the testing of processing relations is performed along with the integration testing. The authors also showed how a SXM model of a *P* system can be obtained and how the NSXM testing approach can be applied to generate conformance test sets for the *P* system.

6. Conclusions and Future Work

Testing aims at finding errors in the tested object and giving confidence in its correct behavior by executing the tested object with selected input values. At present, there are no systematic method and tool that are employed to test Web applications efficiently. The improved traditional methods or a new method appropriate for Web application testing are desired urgently for all the characteristics of Web applications. Since the current testing methods depend primarily on the testers' intuition and experience, the testing of Web applications is regarded as a time-consuming and expensive process. Therefore, a new methodology for Web testing is required imminently to automate the testing.

The features of Web applications make traditional coverage-based [27] or partition-based testing inappropriate (or inadequate) for Web applications. The proposed SXM model can capture information about control flow, data flow, transaction processing and associated usage as well as criticality information. Test cases can be produced by following the states and state transitions in SXM to select individual operations (states) and link them (transitions) together to form overall end-to-end operations.

There is the famous 80 - 20 rule declaring that typical software spends 80 percent of the time executing 20 percent of the code. That means different portions of software are executed with a higher frequency than others. Statistical usage testing, often originally referred to the work by Adams [28], aims to identify these portions and adjusts test suites subjecting more frequently executed parts to more thorough testing. It is often important, however, to ensure the exercise of specific operations of the software irrespective of their usage probabilities. Two examples are operations of high criticality due to potential impacts of a failure and those implemented by new software.

One of the challenging steps planed for further research is to combine SXM-based testing with statistical usage testing for Web application testing. Another critical problem of test generation is adequacy criteria. This problem should also be considered In Web testing. However, this work doesn't cover it. It may be discussed in testing generation approach in the future research.

7. Acknowledgements

This work is financially supported by the National Natural Science Foundation of China (NSFC) under grant No. 61163007, the Jiangxi Provincial Natural Science Foundation of China under Grant No. 2010GQS0048, the Nuclear High Base Special Funds under Grant No. 2010 ZX01045-001-001 and the School Foundation of Jiangxi University of Finance & Economics of China under Grant No. 05072015.

REFERENCES

[1] E. Kirda, M. Jazayeri and C. Kerer, *et al*., "Experiences in Engineering Flexible Web Services," *IEEE MultiMedia*, Vol. 8, No. 1, 2001, pp. 58-65. doi:10.1109/93.923954

[2] E. Hieatt and R. Mee, "Going Faster: Testing the Web Application," *IEEE Software*, 2002, pp. 60-65. doi:10.1109/52.991333

[3] G. A. D. Lucca and A. R. Fasolino, "Testing Web-Based Applications: The State of the Art and Future Trends,"

Information and Software Technology, Vol. 48, No. 6, 2006, pp. 1172-1186. doi:10.1016/j.infsof.2006.06.006

[4] S. Eilenberg, "Automata, Languages and Machines," Vol. A, Academic Press, New York, 1974.

[5] M. Holcombe and F. Ipate, "Correct Systems: Building Business Process Solutions," Springer Verlag, Berlin, 1998.

[6] G. Laycock, "The Theory and Practice of Specification-Based Software Testing," Ph.D. Thesis, Sheffield University, Sheffield, 1993.

[7] E. Kapeti and P. Kefalas, "A Design Language and Tool for X-Machine Specification," In: D. Fotiadis and S. Nikolopoulos, Eds., *Advances in Informatics*, World Scientific, 2000, pp. 134-145.

[8] P. Kefalas, G. Eleftherakis and A. Sotiriadou, "Developing Tools for Formal Methods," *Proceedings of the 9th Panhellenic Conference in Informatic*s, 2003, pp. 625-639.

[9] F. Ipate and M. Holcombe, "An Integration Testing Method That Is Proven to Find All Faults," *International Journal of Computer Mathematics*, Vol. 63, 1997, pp. 159-178. doi:10.1080/00207169708804559

[10] T. S. Chow, "Testing Software Design Modelled by Finite State Machines," *IEEE Transactions on Software Engineering*, Vol. 4, No. 3, 1978, pp. 178-187. doi:10.1109/TSE.1978.231496

[11] F. Ipate and M. Holcombe, "Generating Test Sequences from Non-Deterministic Generalized Stream X-Machines," *Formal Aspects of Computing*, Vol. 12, No. 6, 2000, pp. 443-458. doi:10.1007/s001650070004

[12] X. Jia, H. Liu and L. Qin, "Formal Structured Specification for Web Applications Testing," 2003 *Midwest Software Engineering Conference*, Chicago, 2003.

[13] G. A. Stout, "Testing a Website: Best Practices," A Whitepaper, 2011. http://www.reveregroup.com

[14] Z. S. Qian, "An Approach to Testing Web Applications Based on Probable FSM," 2009 *International Forum on Information Technology and Applications*, Chengdu, 2009, pp. 519-522.

[15] A. Andrews, J. Offutt and R. Alexander, "Testing Web Applications by Modeling with FSMs," *Software Systems and Modeling*, Vol. 4, No. 3, 2005, pp. 326-345. doi:10.1007/s10270-004-0077-7

[16] Z. S. Qian, "Test Case Generation and Optimization for User Session-Based Web Application Testing," *Journal of Computers*, Vol. 5, No. 11, 2010, pp. 1655-1662. doi:10.4304/jcp.5.11.1655-1662

[17] S. Elbaum, S. Karre and G. Rothermel, "Improving Web Application Testing with User Session Data," *The 25th International Conference on Software Engineering*, Portland, 2003, pp. 49-59.

[18] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications," *The 23rd International Conference on Software Engineering*, Toronto, 2001, pp. 25-34.

[19] C. H. Liu, D. C. Kung and P. Hsia, "Object-Based Data Flow Testing of Web Applications," *The First Asia-Pacific Conference on Quality Software*, HongKong, 2000, pp. 7-16.

[20] B. M. Subraya and S. V. Subrahmanya, "Object Driven Performance Testing of Web Applications," *The First Asia-Pacific Conference on Quality Software*, HongKong, 2000, pp. 17-26.

[21] Z. S. Qian, "Towards Testing Web Applications Using Functional Components," *Journal of Software*, Vol. 6, No. 4, 2011, pp. 740-745.

[22] Kourtesis, Dimitrios, Ramollari, Ervin, Dranidis, Dimitris and P. Iraklis, "Discovery and Selection of Certified Web Services Through Registry-Based Testing and Verification," In: Pervasive Collaborative Networks, *IFIP International Federation for Information Processing*, Springer, Boston, pp. 473-482.

[23] F. Ipate and M. Holcombe, "Testing Data Processing-Oriented Systems from Stream X-Machine Models," *Theoretical Computer Science*, Vol. 403, No. 2-3, 2008, pp. 176-191. doi:10.1016/j.tcs.2008.02.045

[24] M. G. Merayo, R. M. Hierons and M. Nunez, "Extending Stream X-Machines to Specify and Test Systems with Timeouts," *The 6th IEEE International Conference on Software Engineering and Formal Methods*, 2008, pp. 210-210.

[25] K. Bogdanov, M. Holcombe, F. Ipate, L. Seed and S. Vanak, "Testing Methods for X-Machines: A Review," *Formal Aspects of Computing*, Vol. 18, No. 1, 2006, pp. 3-30. doi:10.1007/s00165-005-0085-6

[26] F. Ipate and M. Gheorghe, "Testing Non-Deterministic Stream X-Machine Models and P Systems," *Electronic Notes in Theoretical Computer Science*, Vol. 227, 2009, pp. 113-126. doi:10.1016/j.entcs.2008.12.107

[27] B. Beizer, "Software Testing Techniques," 2nd Edition, International Thomson Computer Press, New York, 1990.

[28] E. N. Adams, "Optimizing Preventive Service of Software Products," *IBM Journal of Research and Development*, Vol. 28, No. 1, 1984, pp. 2-14.