A Test Case Generation Method Based on State Importance of EFSM for Web Application

Junxia Guo, WeiWei Wang, Linjie Sun, Zheng Li and Ruilian Zhao
College of Information Science and Technology
Beijing University of Chemical Technology
Beijing, China
Email: gjxia; lizheng; rlzhao@mail.buct.edu.cn

[1] ***Abstract*—Test cases generation is a principal process in web application testing. Most existing methods generate test cases for improving test efficiency mainly from the aspects like minimizing the test case suite, increasing the code coverage, and so on. However, similar with traditional software having important functions, classes or modules, some web states are more vital than others in web applications. It can be thought that those vital web states relatively have higher influence on the performance of web application. So, they should be given more attention in test case generation. In more detail, the importance of web states can be measured from its page contents or topological structures. Meanwhile, as we known Model-based Testing is a kind of widely used approach in automatic test case generation. Therefore in this paper we propose an EFSM based test case generation method considering the importance of web states for web applications. The experimental results show that our methods can deterministically enhance the testing efficiency of web application.**

***Index Terms*—Web application testing; test case generation; Web state importance; EFSM model;**

I. Introduction

With the high speed development of Internet, web applications have become widespread and still increase rapidly. Web applications testing becomes more and more difficult. As an main part of testing, test cases generation is crucial to improve the test efficiency. Unquestionable, good test cases can detect bugs faster. So, test case generation approaches for web applications aim to generate better test cases automatically. There are several kinds of methods with respect to test case generation for web applications, for instance, Capture/Replay-based methods, source code analysis-based methods and model-based methods. Most of those methods generate test cases for improving test efficiency mainly from the aspects like minimizing the test case suite, increasing the code coverage, and so on. However, similar with traditional software having important functions, classes or models[1], some web states are more vital than others in web applications. These web states relatively have higher influence on the performance of web application. They should be tested preferentially. So, the test case generation for web applications should give more attention on these web states.

In order to generate test cases that can cover the important web states, we need to, firstly, find those web states in a web application. As we known, web states of a web application can be described with a graph, such as a state-flow graph, a state-transition graph and so on. Although the importance of nodes can be calculated according to the algorithms of graphic theory, the importance is only reflected from the topology of graph, without referring to the importance of a web page itself. Meanwhile, nowadays, more and more new dynamic page techniques are adopted in web applications in order to get better user experience with the help of client-side scripting or server-side scripting or both of them. Asynchronous JavaScript and XMLHttpRequest $(Ajax)$ is a set of those techniques which can change content dynamically without reloading the entire page, hence can decrease the information transfer between client side and server side. This makes the traditional methods that use a web page as a web state for web application test become not suitable. So, we need to pay more attention to the change of Document Object Model (DOM) structures.

Model-based testing (MBT) is a promising paradigm for test case generation. Therefore, in this paper, we firstly propose a method of building Extended Finite State Machine$(EFSM)$ model based on session data for web applications, in which a state node represents a web state in the format of combining a URL and DOM structure. Then, we measure the importance of states from that of page contents or topological structures, and give an algorithm to evaluate the importance of web states. Finally we present a test case method based on the state importance of EFSM model.

The primary contributions of this paper are as follows:

1) Propose a method for building EFSM model based on session data, which can illustrate a web application more explicitly.
2) Present a test case generation method based on the importance of web states, which considers the aspects of page contents and topological structures.
3) Implement a prototype tool and empirically evaluate our EFSM model building method and test case generation method. The results show that our methods are usable and effective.

The rest of this paper is organized as follows. Section II presents an overview of the related work. Section III describes how to construct an EFSM model for a web application based on its session data in detail. Section IV explains the algorithm

[1]DOI reference number: 10.18293/SEKE2018-177

for evaluating the importance of web states from the aspects of page contents and topological structures. Section V reports the experimental results and analysis. Finally, conclusion and future work are given in Section VI.

II. Related Work

In this paper, we broadly categorise test case generation techniques for web applications into three groups, including Capture/Replay-based techniques, source code analysis-based techniques, model-based techniques. Capture/Replay-based techniques are one of notable trends for the automated test case generation and execution in the area of web application testing in recent years. Currently the most popular capture-replay tools for AJAX testing are Selenium[2], Sahi[3], and Watir[4], which can test DOM-based web applications by capturing events information from user interaction. Such tools need to access the DOM, can assert expected UI behaviour defined by the tester and replay the events[5]. However, a substantial amount of manual efforts are required for testing.

Source code analysis-based techniques generate test cases based on the information which are obtained from the page contents of client-side pages or the code structure of server-side code of web applications. Such as, Wang et al.[6] proposes a static analysis approach for automatic generating test cases for web applications. In this approach, source code is analyzed to extract interfaces which are composed of input parameters with domain information and user navigation map which is composed of all the possible URLs from web application source code. Then through the navigation graph, a set of paths is selected and test cases are generated for each path. Guodong et al.[7] present SymJS, a symbolic framework for both pure JavaScript and client-side web programs. The tool contains a symbolic execution engine for JavaScript, and an automatic event explorer for web pages. Mark Harman et al.[8] introduce three related algorithms and a tool for automated web application testing using Search Based Software Testing $(SBST)$. Their approach starts with a static analysis phase that collects static information to aid the subsequent search based phase and produces a test suite that maximizes branch coverage of the server-side application under test.

Model-based testing(MBT) is an increasingly wide-used approach which has gained much interest in recent years, from academic as well as industrial domain. The idea of MBT is to create and maintain a model that contains the information about the structure and possibly about the desired behaviours of a web application. Test cases are then derived either manually or automatically from the model with algorithms that systematically cover the model using so called selection criteria. Ricca and Tonella[9] proposed a UML model for web application which incorporates static and dynamic aspects of web application and re-interpreted it as a graph. They developed a tool ReWeb, which is used to create the model, and another tool TestWeb, which is used to generate and execute a set of test cases based on the model built by ReWeb. Alessandro Marchetto et al.[10], [11] proposed a state-based testing approach, specially designed to exercise Ajax web applications. The DOM of the page manipulated by the Ajax code is abstracted into a state model. Callback executions triggered by asynchronous messages received from the web server are associated with state transitions. Test cases are derived from the state model based on the notion of semantically interacting events. QI et al.[12] present a combinatorial strategy for full form test. This approach aims to exploring more states and building a complete automated test model for a web application. Miguel[13] proposed a test generation and filtering technique for model-based testing for web applications. A model contains the information of all UI Test Patterns linked with connectors. Each UI Test Pattern contains its specific configurations with the data needed for test execution. Priti Bansal et al.[14] proposed a Model Based Test Case Generation technique, in which the model represents the navigation behavior of a web application. The related information is derived from requirements and low level design. Traversing the model can generate test sequences which can be incorporated with input data to generate test cases later.

III. Method for Building EFSM Model Based-on Session Data

To describe a web application explicitly, we use the combination of URL and DOM structure as a state. In addition, in order to ensure the executability of generated test cases, the detailed information about how to trigger the transitions is also necessary, including the trigger event, preconditions and follow-up actions. EFSM is a widely used model which consists of states and transitions. It is an enhanced model which adds the preconditions of transitions and actions based on Finite State Machine (FSM). The information about preconditions of transitions and follow-up actions can be depicted on EFSM model. Thus we assume that EFSM is suitable for describing web applications.

An EFSM model is formally represented as a 6-tuple (S, S_0, I, V, O, T), where S is a finite set of states, $S_0 \in S$ is an initial state named START, I is a set of input declarations, V is a finite set of internal/context variables, O is a set of output declarations, T is a finite set of transitions. Each member of I is expressed as event(input parameters) meaning event occurs with a list of input parameters. Each member of O is described as action. Each transition $t \in T$ is represented by a 5-tuple $\langle$source(t), target(t), event(t),condition(t), action(t)$\rangle$, where $source(t) \in S$ is the start state of transition t, $target(t) \in S$ is the target state, $event(t) \in I$ is an incentive event or empty, condition(t) is the preconditions performing transition t, and action(t) represents a sequence of actions[15], [16].

The framework of EFSM model building method is shown in Figure1. It is mainly consisted of four processes. $1)$ Get state and transition-related information through crawling client-side. $2)$ Get transition-related information through session data. $3)$ Combine the information of client-side with the session data. $4)$ Build EFSM model. We present the details in following subsections.

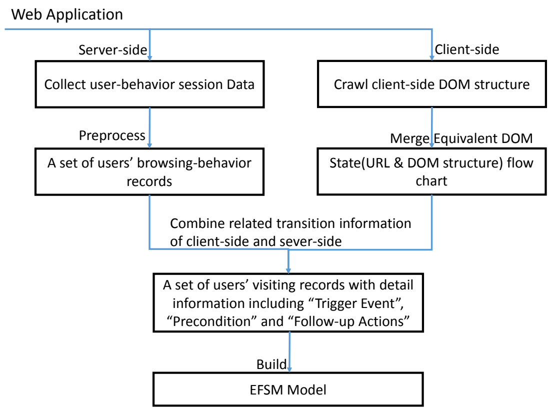

Web Application
Server-side
Client-side
Collect user-behavior session Data
Crawl client-side DOM structure
Preprocess
Merge Equivalent DOM
A set of users' browsing-behavior records
State(URL & DOM structure) flow chart
Combine related transition information of client-side and sever-side
A set of users' visiting records with detail information including "Trigger Event", "Precondition" and "Follow-up Actions"
Build
EFSM Model

Fig. 1. Method Framework of Building EFSM Model

A. DOM Structures and Transition-related Information Collection from Client-side

In this paper, we use a tool named Crawljax[17] to get the DOM structures of a web application. Crawljax works in the mode of depth-first crawl rule. The result includes the states information and the transition-related information which could be got from client-side.

There may exist duplicate states that are triggered by different events and handled by the same script function. Meanwhile, there are states which have the same structure but have different contents, for example different text content. Therefore, in order to avoid state explosion, the above two kinds of states need to be reduced and combined. Firstly, we abstract the DOM structure of all states by filtering the structure-unrelated elements, such as text content, time stamp and so on. Then we judge the similarity of two states by using the edit distance of two DOM structures, which is calculated through Levenshtein[18] method. Finally, we reduce all the same states and combine all the similar states which have high similarity.

B. Session Data Preprocessor

Session data, a kind of log data which can record the original information when a user visits a web application and sends requests. Based on those data, we can get the whole visiting trails of every user. In addition, we can get the transition information through them. In this paper, we collect the session data from server-side. The detailed information of session data mainly includes *User IP*, *Session ID*, *Date & Time*, *Request Event*, *Request URL*, *Referrer URL* and *Responded HTTP Code*.

The preprocessing of session data mainly has three procedures, which are clearing up unusable records, user identification and dividing users' visiting trails. We use session ID to identify different users. There are mainly two kinds of methods for separating user's visiting sequence. One is based on the referrer information. Another is according to the visiting time threshold. In this paper, we united the two kinds of methods as the rule, which can be described as: Record a user's visiting trail based on the page URL referrer. If the interval of the user visiting is larger than 30 minutes, we confirm the user start a new visiting. The time threshold is determined according to the related work[19], [20].

C. Transition-related Information Integration

User-operations of web application may interact with the server-side or not. For example, when deleting a mail in the email system, the selection operation do not interact with server-side while the deletion operation do. In order to get integrated transition-related information, we need to combine the related information got from client-side and server-side.

For example a email deletion operation usually has two steps. Select the check-box to mark the email. Then click the "delete" button to finish the operation. The first step just has client-side information recorded about this transition. The second step has interaction with server-side. The related information will be recorded in session data. The information of those two steps will be combined and recorded to the related users' visiting sequence. The relationship diagram of records in session data could be illustrated as Figure2.

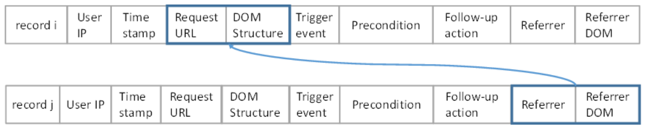


Fig. 2. Relationship Diagram of Records in Session Data

D. Build EFSM Model

As mentioned above, the item of <URL, DOM-structure> is a state in our method. The information of *Trigger Event*, *Precondition*, *Follow-up Action* is integrated transition information. According to the relationships between item <Referrer URL, Referrer DOM> and <Request URL, DOM Structure> shown in Figure2, which describes the transition sequence of states, the EFSM model could be built. The pseudo-code of building EFSM model algorithm is shown in Algorithm 1.

Algorithm 1 EFSM Model Construction

```
Require: integrated user session
Ensure: an EFSM model description file for the web application
 1: $EFSM.init()$; //store the model information
 2: while $session \neq NULL$ do
 3:     while $record \neq NULL$ do
 4:         if $record.state\ NotIn\ EFSM.state$ then
 5:             $EFSM.state.add(record.state)$;
 6:         end if
 7:         $t.init(record)$;//initialize the transition t
 8:         if $t\ NotIn\ EFSM.trans$ then
 9:             $EFSM.trans.add(t)$;
10:         end if
11:     end while
12: end while
```

IV. Test Case Generation Method Considering Web State Importance

We consider the importance of web states mainly from two aspects. One aspect is from the topological structure. Another is from the importance of contents on related web page. Based on the important value of each state, we use traditional Genetic Algorithm(GA) to generate test cases from the EFSM which is built using the method described in SectionIII. We present our method's detail in following subsections.

A. Evaluate Web State Importance

The content of a web page is mainly the text content. The purpose of evaluating page content importance is to derive the importance of the textual content displayed in the current page over other web pages of the web application. For a web application, we can assume that the pages with important content are visited frequently. Thus the web pages containing critical contents are more important for the web application.

The topological structure of web states is the inter-dependency between the related web pages which can be divided into data dependency and link dependency. Data dependency refers to the operation of requesting data from the server during the transition from the current state to a new state. Link dependency refers to a simple jump from the current state to a new state without data interaction.

Therefore, the importance of a web state can be evaluated by combining the importance of the state in topological structure and the related content. In this paper, we firstly calculate content importance of the web page for each state. Then we calculate the state importance directly by introducing the content importance into topological importance calculation.

1) Content importance of a web page: By traversing the state of EFSM model, we can get all web pages for all states. For each web page, we can get a vector model of page content. A keyword vector model reflecting the entire web application is inferred with all pages' vector models. Comparing the vector model of a page and the keyword vector model of the web application, the content importance of the web page can be calculated.

When trigger a transition according to the information of precondition, the textual content of the web page for destination state can be extracted from the HTML code using tag reduction method. The word segmentation operation is performed on the textual content and the number of occurrences of each word is counted. Word segmentation and word frequency constitute a vector model of the page content, which can be formalized as: $State(v) = \{w1 : m1, w2 : m2, , wi : mi\}$. Where w is the word appearing in the page and m is the occurrence frequency of the word. After obtaining the vector models of all pages, a keyword vector model of the web application is inferred with all pages' vector models $Key = \{w1 : n1, w2 : n2, , wi : ni\}$.

The content importance($Con\_Importance$) of a web page can be calculated through comparing the vector model of the page($State(v)$) and the keyword vector model of the web application(Key). The calculation formula is as follows. $Con\_Importance(v) = sin(State(v), Key) = cos\theta = \frac{State(v)\cdot Key}{||State(v)||\cdot||Key||}$

2) State importance evaluation: The state importance evaluating algorithm presented here draws from the PageRank algorithm's logic, whose core idea is that the page with high importance relies on the important pages, and the page relying on high importance pages is of high importance[21].

Including the content importance, we consider 5 factors totally , which are the importance of the states which current state has data or link dependency and those depend on current state. We divide those 5 factors in two attributes: Authority and Hub. Authority refers to the summed importance of the states on which the current state depends. There are mainly three parts to measure authority: data dependency, link dependency and content importance. The calculation formula is as follows, where $Authority(v)$ is the summed importance of the states($includingu, w$) on which the current state(v) depends. $(w, v) \in E, DTR$ means that state v has a data dependency on state w, and $(u, v) \in E, LTR$ means state v has a link dependency on state u. $Importance(w)$ means the state importance of state w. α is a constant more than 1 since we think the data dependency is more important than the link dependency.

$$Authority(v) = \sum_{(w,v)\in E,DTR} \alpha Importance(w) + \sum_{(u,v)\in E,LTR} Importance(u) + Con\_Importance(v)$$

Hub refers to the summed importance of states that depend on the state. It mainly consists of two parts: data dependency and link dependency. The calculation formula is as follows.

$$Hub(v) = \sum_{(w,v)\in E,DTR} \alpha Importance(w) + \sum_{(u,v)\in E,LTR} Importance(u)$$

So a state's importance can be calculated with the formula $Importance(v) = Authority(v) + Hub(v)$.

The algorithm for calculating state importance is described in Algorithm 2. The web application is abstracted into the form of an EFSM model. The initial value of state importance equals the Authority value, which equals to the content importance. The initial Hub value equals 0. In the iterative process, the value of current importance is compared with that of the previous generation. And when the difference is less than threshold ϵ, the iteration ends and the page importance result is obtained. Note that $Importance_t$ refers to an importance value vector of all the web pages in the t_{th} iteration.

Algorithm 2 Web State Importance Calculation

```
Require: $EFSM =< state, trans >$, the threshold $\epsilon$
Ensure: Pages' content importance value
 1: for $v$ in state do
 2:     $Importance(v) = Authority(v), H(v) = 0$
 3: end for
 4: while $||Importance_t - Importance_{t-1}|| <= \epsilon$ do
 5:     for $v$ in state do
 6:         $Authority_t(v) = \sum_{(w,v)\in E,DTR} \alpha Importance_{t-1}(w) + \sum_{(u,v)\in E,LTR} Importance_{t-1}(u) + Con\_Importance_{t-1}(v)$
 7:         $Hub(v)_t = \sum_{(w,v)\in E,DTR} \alpha Importance_{t-1}(w) + \sum_{(u,v)\in E,LTR} Importance_{t-1}(u)$
 8:         $Importance_t(v) = Authority(v)_t + Hub_t(v)$
 9:     end for
10:     $Importance_t = \frac{Importance_t}{||Importance_t||}$
11:     t=t+1
12: end while
```

B. Test Case Generation

There are many test generation approaches for EFSMs. The search-based algorithm is the most commonly used. In this paper, we use GA to generate test cases based on state importance of EFSM model for web applications.

A test case is one transition path on EFSM which can be expressed as a sequence $Ind =< t_1, t_2, , t_i, t_j, , t_n >$. A test case is an individual, and the initial population is randomly generated from the EFSM. Fitness function is to guide the evolution towards optimal solutions. It determines whether an individual can be selected into the next evolution. We design the fitness function from two aspects. One is maximizing the summed importance of the web states in an individual. The other is minimizing the repeated ratio of transitions in an individual. The fitness function can be formalized as follows.

$$fitness = \sum_{i=0}^{n} Importance(si) \cdot \frac{||UniTrans||}{||Ind||}$$

In this formula, $Importance(s_i)$ refers to the importance of the web state s_i, $||UniTrans||$ is the number of not-repeated transitions and $||Ind||$ is the number of transitions in this individual Ind.

Genetic operators include selection, crossover and mutation. Selection is based on the fitness of the individual, which means the individual of greater fitness has larger probability to be selected as the parent. Crossover operator in our method is single-point crossover. Mutation is applied to alter gene values in an individual.

V. Experiment

In this section, we firstly validate the feasibility of our modelling method. Then we generate test cases from the EFSM model guided by the state importance to demonstrate the benefit of our method. To assess the effectiveness of our method, we conduct case studies on two web applications. The following research questions motivate our experiments.

RQ1: Is our session data based modelling method effective and feasible?

RQ2: Do the test cases which are generated guided by state importance from the EFSM, cover the important web states earlier?

A. Experimental Subjects

In the experimental studies, we use an open source online Book Store[22] and the laboratory management system (DBLab) developed by our group as the subjects to evaluate the validity and effectiveness of our method. The lines of code(LOC) of DBLab and Book store is 10162 and 6304 respectively. Book store allows users to add and remove books from a cart, login and register new users and so on. DBLab is a laboratory management system that includes user registration and login, group meeting management and viewing, user account management, library management, file management, student forums, data sharing and other functional modules.

B. EFSM Model Evaluation

The EFSM model is abstracted based on the Session data recorded on the server-side when the users access the web application. It may not be sufficient for modelling the web application. The integrity of the EFSM model is directly related to the Session data. In order to analyze the dependency between the model and the Session data, and judge whether the EFSM model is integral, we perform the following experiment.

Taking the DBLab as an example, according to the record time of session data, the EFSM model is established with session data from one month, two months, three months, and four months respectively. Finally, the EFSM model of the web application is manually analyzed and checked to verify if there are missing states or transitions. The experimental results are as follows.

TABLE I
Statistics of DBLab generates EFSM by month

Item	records N. in Session	state N. on EFSM	Transition N. on EFSM
June	954	20	40
June to July	1583	24	54
June to August	1942	24	54
June to September	2814	26	61
Manually checked	–	28	65

It can be seen from the Table I, after modelling the web application manually, the EFSM model increased two states and four transitions. Through analyzing the extra states and transitions, we find that these functions of DBLab are not daily use type. Based on the above analysis, we can see that the proposed EFSM model construction method for web applications in this paper has a great dependence on Session data. However, if the users' session records for web applications reach certain level of saturation, the EFSM model established based on the session data can basically reflect all the functions of the web application and realize the complete description for the web application. So our session data based modelling method is effective and feasible$(RQ1)$.

The detail information of EFSM models which are built through our method for two test subjects is, $1)$ the number of states and transitions of the DBLab's model is 28 and 65 respectively; $2)$ the number of states and transitions of the BookStore's model is 9 and 36 respectively.

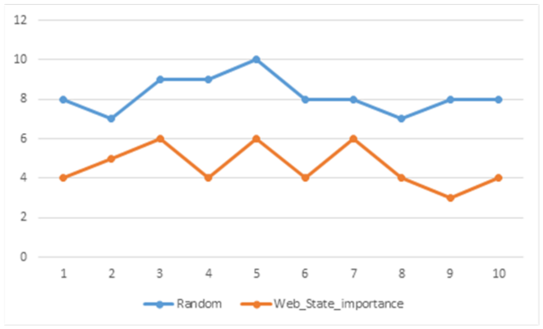
The EFSM model of two test subjects can be found at http://research.cs.buct.edu.cn/guo/files/EFSM-BS.pdf and http://research.cs.buct.edu.cn/guo/files/EFSM-DBLab.pdf.

C. Test Case Generation Method Evaluation

The purpose of the test case generation method proposed in this paper is to prioritize those nodes with high importance to be tested earlier. Because that such nodes have high error propagation capability and are more likely to bring negative affect in the web applications. We select the three most important nodes from the BookStore and DBLab as the high importance nodes according to the node importance value. The three high importance nodes in the BookStore are State1, State2, and State8. The three high importance nodes in the DBLab are State16, State18, and State22.

We use our method and the random method to generate test cases from the EFSMs of the two applications. The test suite size(population size) are set to 10 (BookStore) and 20 (DBLab). The goal of the test suite is to cover all the transitions. Our test case generation method and random method were executed 10 times repeatedly. The results is that, in the BookStore, the number of test cases generated by our method is 4.6 to cover the three high importance nodes on average, and the number of test cases generated by random method is 8.2 on average. In DBLab, the number of test cases generated by our method is 5.2 to cover the three high importance nodes on average, and the number of test cases generated by random method is 15.8 on average. The importance of test suites generated by our method and random method for ten times is shown in Figure 3(H : experimental number, V: number of generated test cases).

[BookStore]



[DBLab]

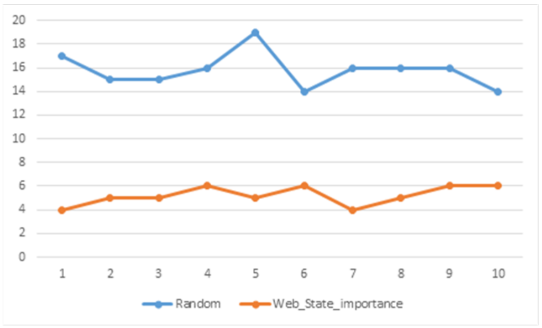


Fig. 3. The importance of test suites comparison

As the Figure 3 shows, under the premise of full transitions coverage, the test suite generated by our method has a higher importance value than the randomly generated test suite. At the same time, our test suite can cover important nodes when executing a small number of test cases. It can be seen that the method proposed in this paper can ensure that nodes with high importance to be tested firstly($RQ2$).

VI. Conclusion an Future Work

In this paper, we proposed an EFSM model constructing method for web applications based on session data, which can address the accurate description problem of web applications, and an algorithm to evaluate state importance of the EFSM model. In addition, we give a test case generation method based on the state importance of EFSM model, which can ensure the important web states can be tested first and then improve the efficiency of web application test.

The experimental results show that our methods can modeling web application well and the generated test case an cover the important web state earlier.

As the future work, firstly we will try to modify the EFSM model constructing method. Because that the method proposed in this paper need a sufficient amount of session data to detect the states and transitions. Secondly, we would like to design other algorithms for evaluating state importance by introducing other items, for example the number of variables.

Acknowledgment

The work described in this paper is supported by the National Natural Science Foundation of China under Grant No.61702029, No.61672085 and No.61472025.

References

[1] M. Hammad, M. L. Collard, and J. I. Maletic, "Measuring class importance in the context of design evolution," in *IEEE International Conference on Program Comprehension*, 2010, pp. 148–151.
[2] "Selenium," https://www.seleniumhq.org/.
[3] "Sahi," http://sahipro.com/.
[4] "Watir," http://watir.com/.
[5] Y. F. Li, P. K. Das, and D. L. Dowe, "Two decades of web application testinga survey of recent advances," *Information Systems*, vol. 43, no. C, pp. 20–54, 2014.
[6] M. Wang, J. Yuan, H. Miao, and G. Tan, "A static analysis approach for automatic generating test cases for web applications," in *International Conference on Computer Science and Software Engineering*, 2008, pp. 751–754.
[7] G. Li, E. Andreasen, and I. Ghosh, "Symjs: automatic symbolic testing of javascript web applications," in *The ACM Sigsoft International Symposium*, 2014, pp. 449–459.
[8] N. Alshahwan and M. Harman, "Automated web application testing using search based software engineering," in *Ieee/acm International Conference on Automated Software Engineering*, 2011, pp. 3–12.
[9] F. Ricca and P. Tonella, "Analysis and testing of web applications," in *International Conference on Software Engineering*, 2001, pp. 25–34.
[10] A. Marchetto, P. Tonella, and F. Ricca, "State-based testing of ajax web applications," in *International Symposium on Search Based Software Engineering*, 2009, pp. 3–12.
[11] A. Marchetto and P. Tonella, "Using search-based algorithms for ajax event sequence generation during testing," *Empirical Software Engineering*, vol. 16, no. 1, pp. 103–140, 2011.
[12] X. F. Qi, Z. Y. Wang, J. Q. Mao, and P. Wang, "Automated testing of web applications using combinatorial strategies," *Journal of Computer Science and Technology*, vol. 32, no. 1, pp. 199–210, 2017.
[13] A. M. Torsel, "Automated test case generation for web applications from a domain specific model," in *Computer Software and Applications Conference Workshops*, 2011, pp. 137–142.
[14] P. Bansal and S. Sabharwal, "A model based approach to test case generation for testing the navigation behavior of dynamic web applications," in *Sixth International Conference on Contemporary Computing*, 2013, pp. 213–218.
[15] A. S. Kalaji, R. M. Hierons, and S. Swift, *An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models*. Butterworth-Heinemann, 2011.
[16] A. S. Kalaji and Hierons, "Generating feasible transition paths for testing from an extended finite state machine (efsm)," in *International Conference on Software Testing, Verification, and Validation Workshops*, 2010, pp. 230–239.
[17] "Crawljax," http://crawljax.com.
[18] A. Mesbah and A. V. Deursen, "Migrating multi-page web applications to single-page ajax interfaces," in *European Conference on Software Maintenance and Reengineering*, 2007, pp. 181–190.
[19] L. D. Catledge and J. E. Pitkow, "Characterizing browsing strategies in the world-wide web," in *International World Wide Web Conference*, 1995, p. 10651073.
[20] J. Guo, C. Gao, N. Xu, G. Lu, and H. Han, "Analyzing query trails and satisfaction based on browsing behaviors," in *Web Information System and Application Conference*, 2014, pp. 107–112.
[21] L. L, D. Chen, X. L. Ren, Q. M. Zhang, Y. C. Zhang, and T. Zhou, "Vital nodes identification in complex networks," *Physics Reports*, vol. 650, pp. 1–63, 2016.
[22] "Book store," http://gotocode.com/.