Towards Testing Web Applications Using Functional Components

Zhongsheng Qian
School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, China
Email: changesme@163.com

***Abstract*—With the prevalence of Internet, the rapid development of component, middleware and Web services, and the wide application of the Web, the reliability and quality assurance of Web applications have become a very critical problem and a hot research topic. To ensure the security and reliability of Web applications, Web testing is one of the most effective methods. A Web application is divided into a set of functional components, each of which offers a certain kind of Web service. A Component Dependency Diagram (CDD) is employed to represent the structural relationship among the functional components; FSMs are used to represent their behaviors and the composition of FSMs to represent their interactions. It presents two test criteria including complete executing sequence coverage and component complete executing sequence coverage. A detail test process is illustrated according to the proposed test criteria.**

***Index Terms*—Web application; functional component; test case**

I. INTRODUCTION

Web applications are among the fastest growing classes of software systems today. These Web applications are being used to support a wide range of important activities: business transactions such as product sale and distribution, scientific activities such as information sharing and proposal review, and medical activities such as expert system-based diagnoses. Given the importance of such applications, 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]. 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. We propose an approach to testing Web applications based on functional components according to the functional requirements of Web applications.

The remainder of this paper is organized as follows. Section II illustrates how to model a Web application according to functional components. Section III presents a modeling approach to dealing with the interactions of functional components. Section IV details the test case generation process. A survey of related work is given in Section V. Section VI draws some concluding remarks and highlights the future work.

II. FUNCTIONAL COMPONENTS FOR A WEB APPLICATION

A computation unit that offers a certain kind of Web service is regarded as a functional component. In Web applications, a functional component may be an individual Web page, a software module, or collections of Web pages and software modules. A software module may be a Java applet, an ActiveX control, or a Java Bean. Web application testing focuses on the relationship among their computation units (functional components). We presume that the computation units are adequately tested before proceeding with any further test in this work. Two important concepts are given as follows.

- *Functional component*. A *functional component* is a completely-encapsulated unit, which can accomplish a function of the Web application independently. It consists of component name and two types of interfaces (*Input* and *Output*). Data are sent to *Output* or received from *Input* through different actions. A functional component may be composed of several other fine-grained functional components. In the view of users, a Web application is regarded as a black box, which can be divided into a set of components according to its functions.
- *Component connector*. A *component connector* is an abstract mechanism of communication, mediation or coordination among components. It is a bridge among components.

We take a miniature *News Publishing* Web application as an example. The Web application is divided into a set of functional components according to the functional requirements. The Component Dependency Diagram (CDD) is shown in Fig. 1. The dependent relationship among components is represented by component connectors. In Fig. 1, each arrow shows that if an event is triggered by one component, then the component it depends on must be executed first. For example, *Login*

doi:10.4304/jsw.6.4.740-745

must be run first to guarantee *PublishNews* to work normally. The *PublishNews* consists of two functional components: *EditNews* and *NewsEditor*. *EditNews* is a component, which converts the news that the user edits into the corresponding format and passes the result as an output to *NewsEditor*. Then, *NewsEditor* adds the title, date and the writer's information to this news. While *NewsEditor* transfers nothing to *EditNews*.

The *News Publishing* Web application can be defined as a set of functional components. The top level set is *WebSet*={*PublishNews*, *UserManager*, *Login*, *Main*, *Exit*}; the *UserManagerSet* is a set of *UserManager*, i.e., *UserManagerSet*={*ChangePassword*, *ChangeUserName*}; the *PublishNewsSet* is a set of *PublishNews*, i.e., *PublishNewsSet*={*EditNews*, *NewsEditor*}.

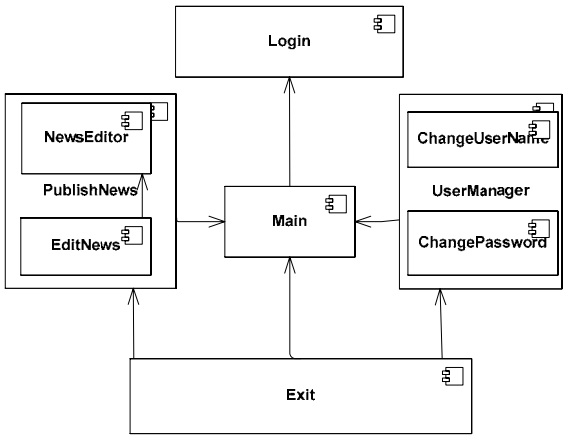

Figure 1. An example component dependency diagram

A functional component communicates with other components asynchronously and synchronously by passing messages that exchange data and activity state information. The interactive relationship between two components can be expressed by a Component Relation Table (CRT), denoted as $K=\{C, R\}$, where C is a set of all the functional components; $R=\{R_1, R_2, R_3\}$ is a set of relations between any two specified functional components with R_1, R_2, R_3: $C \leftrightarrow C$, and

- $(c1, c2) \in R_1$ iff $c1 \in c2$;
- $(c1, c2) \in R_2$ iff $\exists c \in C \cdot c1 \in c \wedge c2 \in c$;
- $(c1, c2) \in R_3$ iff $(c1, c2) \notin R_1 \wedge (c1, c2) \notin R_2$.

In Table I, the relationship among the components of *News Publishing* Web application is clearly described, as can also guide the test case generation. According to the CDD and CRT given above, we can construct its formal testing model based on FSM (without considering the actions of components, see section III for the details of the interactions of components), as is shown in Fig. 2.

TABLE I. THE COMPONENT RELATION TABLE

Component A	Components B	Relation	Is Interactive?
Login	*Main*	*R3*	No
Login	*Exit*	*R3*	No
ChangePassword	*ChangeUserName*	*R2*	No
NewsEditor	*EditNews*	*R2*	Yes
PublishNews	*UserManager*	*R3*	No
EidtNews	*NewsEditor*	*R2*	Yes
… …	… …	… …	… …

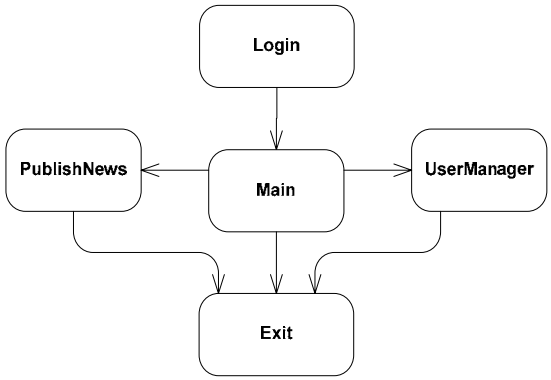

Figure 2. The top FSM for the example component dependency diagram

The FSMs for the inner components of *PublishNews* and *UserManager* are shown in Fig. 3 and Fig. 4 respectively.

Figure 3. The FSM for the inner components of *PublishNews*

Figure 4. The FSM for the inner components of *UserManager*

These two models are some different, for there is dependent relationship among the inner components of *PublishNews* (i.e., if we want to complete the execution of *PublishNews*, then *NewsEditor* must be executed before *EditNews*), while there is no dependent relationship among the inner components of *UserManager* (i.e., we can selectively execute *ChangeUserName* or *ChangePassword*). However, the interactions among components exist in all these two situations.

III. MODELING THE INTERACTIONS OF FUNCTIONAL COMPONENTS

A Web application is a highly interactive system, which can be regarded as a set of interactive functional components. The functional components interact with each other according to their *actions*. When the output action of one functional component corresponds to the input action of another, it shows that an interaction occurs. The behaviors and their interactions of functional components can be described by FSMs. An FSM is a quintuple (S, Act, δ, I, H), where

- S is a finite set of states;
- Act is a finite set of actions;
- $\delta \subseteq S \times \Sigma \times S$ is a finite set of transitions;
- $I \subseteq S$ is a nonempty set of initial states and
- H is a tree-like structure corresponding to a hierarchy of functional component IDs.

Additionally, we define $\Sigma = ((C \cup \{-\}) \times Act \times (C \cup \{-\})) \setminus (\{-\} \times Act \times \{-\})$, where $C = \{c \mid c$ is the name of a functional component, c occurs in $H\}$ and the symbol "–" specifies no component. The symbol $(-, b, B) \in \Sigma$ represents that the component B receives an action b as an input; the symbol $(A, a, -) \in \Sigma$ represents that the component A sends an action a as an output; the symbol $(A, a, B) \in \Sigma$, called internal symbol, represents that the component A sends an action a as an output, and synchronously the component B receives the action a as an input.

The FSM of *PublishNews* is shown in Fig. 5 and the actions table of functional components is given in Table II.

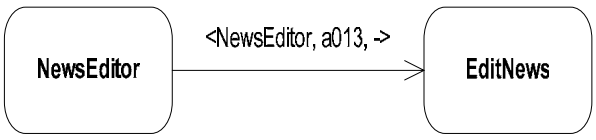

Figure 5. The FSM for the interaction between *NewsEditor* and *EditNews*

According to the constructing method in Fig. 5 and the component actions table in Table 2 (note that, each action of components corresponds to an actual operation), some instances of FSMs for component interactions can be given as follows:

- The FSM for *NewsEditor* with only one *Output* action can be specified as:
  FSM_NewsEditor = ({q0}, {a013}, {(q0, (*NewsEditor*, a013, -), q0) }, {q0}, {*NewsEditor*});
- The FSM for *Login* with only one *Input* action can be specified as:
  FSM_Login = ({q0}, {a001}, {(q0, (-, a007, *Login*), q0)}, {q0}, {*Login*});
- The FSM for *EditNews* with one *Input* and one *Output* action can be specified as:
  FSM_EditNews = ({q0, q1}, {a014, a015}, {(q0, (-, a014, *EditNews*), q0), (q0, (*EditNews*, a015, -), q1)}, {q0}, {*EditNews*}).

TABLE II. COMPONENT ACTIONS TABLE

Component Name	Component ID	Component Action
Login	1	<*Login*, a001, ->
Main	2	<-, a002, *Main*>, <*Main*, a003, ->, <*Main*, a004, ->, <*Main*, a005, ->, <*Main*, a006, ->
PublishNews	3	<-, a007, *PublishNews*>, <*PublishNews*, a008, ->, <*PublishNews*, a009, ->
UserManager	4	<-, a010, *UserManager*>, <*UserManager*, a011, ->, <*UserManager*, a012, ->
NewsEditor	5	<*NewsEditor*, a013, ->
EditNews	6	<-, a014, *EditNews*>, <*EditNews*, a015, ->
ChangePassword	7	<*ChangePassword*, a016, ->
ChangeUserName	8	<*ChangeUserName*, a017, ->
.....	…..	……

IV. GENERATING TEST CASE

The behaviors of components can be manifested by the services provided for the outside through their interfaces (input actions), the called services of other components (output actions) and the execution of inner operations (internal actions). We can generate component interaction test sequences satisfying EC (Each Choice) coverage, t-wise (t-way) coverage or their combinations, etc. [4-5]. The ideal way of testing the component interactions is to use the AC (All Combinations) strategy [6] to generate all test cases. The AC strategy generates all possible combinations of interesting values of the input parameters. However, it takes too much. So, we often choose the least test sequences to cover most component interactions as possible.

Usually, there exists some sequential relation or constraint among functional components. So, when testing these components, we must take their executing order into account. In executing the functional components, the combinatorial orders may be several. Take the components A, B and C for example, if A is executed first, then B and C are executed, we can get an executing sequence $A \rightarrow B \rightarrow C$, denoted by $<A, B, C>$; or the executing sequences may be $B \rightarrow A \rightarrow C$ or $C \rightarrow A \rightarrow B$ and so on, i.e., totally 6 cases exist. We regard abstract test cases as the executing sequences of components. Therefore, the following two important concepts are given.

Definition 1. An *executing sequence* is a sequence of functional components that are executed in a certain order.

Definition 2. A *complete executing sequence* is an executing sequence that starts from the initial state to the

end state.

According to Definition 2, we present a test criterion: *complete executing sequence coverage*.

Definition 3. A test set *TS* satisfies *complete executing sequence coverage*, if and only if for any complete executing sequence *seq*, there is one test case *t* in *TS* and *t* passes *seq*.

Example 1. In the top FSM for the example CDD in Fig. 2, *Login* is in an initial state, and *Exit* is in an end state. The test set *TS*={<*Login*, *Main*, *Exit*>, <*Login*, *Main*, *PublishNews*, *Exit*>, <*Login*, *Main*, *UserManager*, *Exit*>} satisfies *complete executing sequence coverage*.

According to Table 2, running each test case in *TS* means that each action in the components in the test case is triggered in a given order.

In Example 1, *PublishNews* is a composite component, which consists of *NewsEditor* and *EditNews*, whose relation type is *T2* according to Table 1, and there is an interaction between *NewsEditor* and *EditNews* (see Fig. 3). Then, <*Login*, *Main*, *PublishNews*, *Exit*> can be represented as <*Login*, *Main*, <*NewsEditor*, *EditNews*>, *Exit*> (i.e., <*Login*, *Main*, *NewsEditor*, *EditNews*, *Exit*>), but not as <*Login*, *Main*, <*EditNews*, *NewsEditor*>, *Exit*>, for *NewsEditor* must be executed before *EditNews* according to Fig. 3; *UserManager* is also a composite component, which consists of *ChangeUserName* and *ChangePassword*, whose relation type is also *T2* according to Table 1. However, there is not an interaction between *ChangeUserName* and *ChangePassword* (see Fig. 4). Then, <*Login*, *Main*, *UserManager*, *Exit*> can be represented as either <*Login*, *Main*, <*ChangeUserName*>, *Exit*> or <*Login*, *Main*, <*ChangePassword*>, *Exit*>.

Therefore, *TS* is partitioned into two test sets, which are:

TS1={<*Login*, *Main*, *Exit*>, <*Login*, *Main*, <*NewsEditor*, *EditNews*>, *Exit*>, <*Login*, *Main*, <*ChangeUserName*>, *Exit*>} and

TS2={<*Login*, *Main*, *Exit*>, <*Login*, *Main*, <*NewsEditor*, *EditNews*>, *Exit*>, <*Login*, *Main*, <*ChangePassword*>, *Exit*>}.

Both *TS1* and *TS2* satisfy *complete executing sequence coverage*, however, whichever can not test all the components. For example, *ChangePassword* can not be tested in *TS1,* while *ChangeUserName* can not be tested in *TS2*. So, to cover all the *complete executing sequences*, and all the components are passed at least once, the following test criterion is proposed.

Definition 4. A test set *TS* satisfies *component complete executing sequence coverage*, if and only if for any complete executing sequence *seq*, there is one test case *t* in *TS*, *t* passes *seq* and every component is passed at least once.

Example 2. *TS12* = *TS1* ∪ *TS2* = {<*Login*, *Main*, *Exit*>, <*Login*, *Main*, <*NewsEditor*, *EditNews*>, *Exit*>, <*Login*, *Main*, <*ChangeUserName*>, *Exit*>, <*Login*, *Main*, <*ChangePassword*>, *Exit*>} is one of the test sets, which satisfy *component complete executing sequence coverage*.

Additionally, 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:

- [*c1*, *c2*], which represents that either *c1* or *c2* is selected to execute each time;
- (*c1*, *c2*), which represents that both *c1* and *c2* are executed in any order;
- <*c1*, *c2*>, which represents that both *c1* and *c2* are executed and *c1* is executed before *c2* (as shown above).

For example, the test set *TS12* can be represented as {<*Login*, *Main*, *Exit*>, <*Login*, *Main*, [<*NewsEditor*, *EditNews*>, [*ChangeUserName*, *ChangePassword*]], *Exit*>}, which is a reduced way.

V. RELATED WORK

A Web application is a very complex, distributed, multi-tier, interactive system, which provides a brand-new way for users to deploy software application. The isomerism, dynamics, diverse connections, variant control flows, and rapid development and deployment of Web applications have brought the new challenge for their testing. 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 application testing is required imminently to automate the testing.

A number of Web testing techniques for Web applications have been already proposed [7-20], each of which has different origins and pursues different test goals for dealing with the unique characteristics of Web applications.

Subraya and Subrahmanya [21] presented object driven performance testing. They illustrated a new testing process that employs the concept of decomposing the behavior of a Web application into testable components. Different from theirs, our approach decomposes a Web application according to its functional requirements, not its behavior.

Andrews, et al. [22] 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. [23] 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 [24] 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. [25] 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 [24-25] 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. While our approach concerns mainly the functional aspect towards a black-box testing (a functional test of some sort) at a functional level of abstraction.

Lucca and Fasolino [3] surveyed Web application testing. 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.

VI. CONCLUSIONS AND PERSPECTIVES

As we all know, software testing in general and Web application testing in particular are knowledge-driven, labor intensive activities, which require the testers with quite experiences and professional abilities, and also need a systematical way to guide the testing process.

This work describes a functional component-based approach to generating test cases for Web applications described by CDDs. A Web application is assumed to be composed of interacting functional components. It employs an FSM to describe each component behavior and the composition of FSMs to depict the interacting actions. Two test criteria are presented, according to which the test generation process is illustrated.

The next step is to develop a prototype to automate the testing process and evaluate the approach to dividing a Web application into a sequence of interacting functional components.

ACKNOWLEDGMENT

This work is supported by the Science and Technology Program of the Education Department of Jiangxi Province of China under Grant No. GJJ10120, and the Jiangxi Provincial Natural Science Foundation of China under Grant No. 2010GQS0048. The authors wish to thank the anonymous reviewers for their detailed and helpful suggestions.

REFERENCES

[1] E. Kirda, M. Jazayeri and C. Kerer, et al., "Experiences in Engineering Flexible Web Services", *IEEE MultiMedia*, vol.8, no.1, pp. 58-65, 2001.

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

[3] G. A. D. Lucca, A. R. Fasolino, "Testing Web-based Applications: The State of the Art and Future Trends", *Information and Software Technology*, no.48, pp. 1172-1186, 2006.

[4] A. W. Williams, R. L. Probert, "A Measure for Component Interaction Test Coverage", The ACSI/IEEE International Conference on Computer Systems and Applications, Beirut, Lebanon, June 2001, pp. 304-311.

[5] M. Grindal, J. Offutt and S. F. Andler, "Combination Testing Strategies: A Survey", GMU Technical Report ISE-TR-04-05, July 2004.

[6] M. Grindal, B. Lindstrom and J. Offutt, et al., "An Evaluation of Combination Strategies for Test Case Selection", HS-IDA-TR-03-001, Department of Computer Science, University of Skovde, Sweden, 2003.

[7] R. Hower. Web Site Test Tools and Site Management Tools. Software QA and Testing Resource Center, http://www.softwareqatest.com/qatweb1.html. Accessed on Mar. 2011.

[8] Huaikou Miao, Shengbo Chen, Zhongsheng Qian, "A Formal Open Framework Based on Agent for Testing Web Applications", The 3rd International Conference on Computational Intelligence and Security (CIS' 2007), IEEE CS, pp. 281-285.

[9] William G. J. Halfond, Saswat Anand, Alessandro Orso, "Precise Interface Identification to Improve Testing and Analysis of Web Applications", The 18th International Symposium on Software Testing and Analysis, ACM, NY, USA, 2009.

[10] Liping Li, Huaikou Miao, Zhongsheng Qian, "A UML-Based Approach to Testing Web Applications", 2008 International Symposium on Computer Science and Computational Technology, Shanghai, China, 2008, pp. 397-401.

[11] William G. J. Halfond, Alessandro Orso, "Improving Test Case Generation for Web Applications Using Automated Interface Discovery", The 15th ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2007.

[12] Liping Li, Zhongsheng Qian, Tao He, "Test Purpose-Based Test Generation for Web Applications", The First International Conference on Networked Digital Technologies, Ostrava, The Czech Republic, 2009, pp.251-256.

[13] M. Utting, A. Pretschner, B. Legeard, "A Taxonomy of Model-based Testing", Technical report, Department of Computer Science, University of Waikato, New Zealand, April 2006.

[14] Zhongsheng Qian, "Test Case Generation and Optimization for User Session-based Web Application Testing", *Journal of Computers*, 2010, 5(11): 1655-1662.

[15] H. Reza, K. Ogaard, and A. Malge, "A Model based Testing Technique to Test Web Applications Using Statecharts", In ITNG' 08, Washington, DC, USA, 2008, pp. 183-188.

[16] Zhongsheng Qian, "An Approach to Testing Web Applications Based on Probable FSM", 2009 International Forum on Information Technology and Applications. Chengdu, China, 2009, pp.519-522.

[17] P. Tonella, F. Ricca, "A 2-layer Model for the White-box Testing of Web Applications", International Workshop on Web Site Evolution, Chicago, Illinois, 2004, pp. 11-19.

[18] Zhongsheng Qian, "Testing Component-based Web Applications Using Component Automata", 2009 WASE International Conference on Information Engineering.

IEEE CS, Taiyuan, China, July, 2009, pp.455-458.
[19] A. Belinfante, L Frantzen, and C. Schallhart, "Tools for Test Case Generation", *Model-Based Testing of Reactive Systems*, LNCS 3472, Springer, 2005, pp. 391-438.
[20] Huaikou Miao, Shengbo Chen, Huanzhou Liu, Zhongsheng Qian, "An Approach to Generating Test Cases for Testing Component-based Web Applications", Workshop on Intelligent Information Technology Application (IITA' 2007), IEEE CS, Zhang Jiajie, China, Dec. 2-3, 2007, pp. 264-269.
[21] B. M. Subraya, S. V. Subrahmanya, "Object Driven Performance Testing of Web Applications", The First Asia-Pacific Conference on Quality Software, HongKong, China, Oct. 2000, pp. 17-26.
[22] A. Andrews, J. Offutt and R. Alexander, "Testing Web Applications by Modeling with FSMs", *Software Systems and Modeling*, vol.4, no.3, pp. 326-345, Aug. 2005.
[23] S. Elbaum, S. Karre and G. Rothermel, "Improving Web Application Testing with User Session Data", The 25th International Conference on Software Engineering, Portland, Oregon, May 2003, pp. 49-59.
[24] F. Ricca, P. Tonella, "Analysis and Testing of Web Applications", The 23rd International Conference on Software Engineering, Toronto, Ontario, Canada, 2001, pp.25-34.
[25] 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, China, Oct. 2000, pp. 7-16.

Qian Zhongsheng was born in 1977. He received his M.S. degree from Nanchang University of China in 2002, and Ph.D in Computer Science at Shanghai University of China in 2008. He has also been with the School of Information Technology at Jiangxi University of Finance & Economics of China since 2002. His current research interests include software engineering and Web application testing, etc.