

An approach for dynamic web application testing using MBT

Vikas Panthi¹ · Durga Prasad Mohapatra¹

Received: 30 January 2016/Revised: 30 March 2017/Published online: 15 June 2017

© The Society for Reliability Engineering, Quality and Operations Management (SREQOM), India and The Division of Operation and Maintenance, Lulea University of Technology, Sweden 2017

Abstract Nowadays, web applications play a significant role in the success of the business. Web applications have grown very fast in the current market. They bring new challenges for the researchers in the area of testing, such as heterogeneous representation, dynamic behavior, data flow mechanism, control flow mechanism and some more issues relevant to web applications. This paper presents an approach for model-based dynamic web application testing. This approach considers server side scripting language to test the functional requirements of the web applications. In this approach, first the implicit class object tags are extracted from JSP pages, and the JSP Flow Graph (JFG) is constructed for tracing the requirements using the proposed algorithm JTSG. Then, the test scenarios and concrete test cases are generated for the given web application.

Keywords Test scenarios · Dynamic web application · Model based testing (MBT) · Dynamic web application testing · Implicit class object

1 Introduction

The web has a significant impact on all aspects of our society, from business, education, government, entertainment, industry, to our personal lives. In the software

industry, there is a strong trend towards replacing desktop applications with web applications (Booch and Jacobson 2005; Hall and Brown 2008; Conallen 2002). Nowadays, most of the technologies are new for developing web applications. A web application typically consists of Java Server Page (JSP), Servlets, Javascript, Hyper Text Markup Language (HTML), and Cascading Style Sheet (CSS) for Graphical User Interface (GUI) design (Hall and Brown 2008; Conallen 2002). In a web application, a number of components are integrated (Ricca and Tonella 2000, 2001; Liu 2004; Fujiwara et al. 2011; Kung et al. 2000). These components are (i) web browser (client) (ii) web application server and (iii) database server.

Web applications are popular because they are easy to use and maintain (Booch and Jacobson 2005; Hall and Brown 2008; Conallen 2002). The main advantages of adapting the web for developing software products include. (1) No installation costs, (2) Universal access from any machine connected to the Internet, (3) Platform independent and (4) Automatic upgrade with new features for all users, (5) Independent of the type of the browser in the client machine. Normally, the cost of the web application is relatively less as compared to desktop applications.

Testing these web applications technology is very much important. Web-based testing is categorized into two types (Ricca and Tonella 2000, 2001): (a) Static web application testing. (b) Dynamic web applications testing.

There are many difficulties in testing web applications. First, web applications are distributed through a client/server architecture, with (asynchronous) HTTP request/response calls to synchronize the application state. Second, they are heterogeneous i.e. web applications possess their unique features, such as dynamic behavior, heterogeneous representation, and novel data handling mechanism. These

✉ Vikas Panthi
vpanthi@gmail.com

Durga Prasad Mohapatra
durga@nitrkl.ac.in

¹ Department of Computer Science and Engineering, National Institute of Technology, Rourkela, India

characteristics support to the successful deployment of a web application. There are many techniques to test web applications to ensure better quality in application development. Model-based testing is one of them.

This paper presents a model based testing (MBT) approach for web application testing. Our approach performs functional testing. The proposed approach considers Java Server Pages (JSP) for dynamic web application testing. This technique first extracts the implicit class objects in JSP. Then it constructs the JSP Flow Graph (JFG). After that, JTSG algorithm is applied to generate test scenarios. Finally, executable test cases are developed using the generated test scenarios.

The rest of the paper is structured as follows: Sect. 2 presents how we can apply MBT approaches for dynamic web application testing and advantages of MBT. Section 3 provides the preliminary concepts on web applications and testing. Section 4 presents the proposed approach for generating test scenarios for dynamic web applications. Section 5 describes working of the proposed approach by taking five case studies. Section 6 provides the Comparison with related works. Section 7 concludes the proposed approach.

2 Model based testing of dynamic web pages

There are many approaches available for dynamic web applications testing such as: Non functional search-based testing (Afzal et al. 2008), SOA based Testing (Neto et al. 2011), Requirement specification Based Testing (Barmi et al. 2011), Product lines testing (Neto et al. 2011; Engström and Runeson 2011), GUI Testing (Banerjee et al. 2013), Search Based non functional testing (Afzal et al. 2009), Model based testing (Memon and Nguyen 2010), Formal testing of web services, Search based test-case generation (Ali et al. 2010), Regression test selection techniques (Engström et al. 2010), Combinatorial testing (Grindal et al. 2005), Mutation testing (Jia and Harman 2011), etc. In this section, we discuss on Model based testing. Model-based testing is a relatively new technology to test software. A model that describes the desired behaviour of the system under test (SUT) is the key point in model-based testing. The desired behaviour is often specified in the software requirement specification (SRS) document of the SUT. Model-based testing goes beyond automated testing because it algorithmically generates the specified number of test cases based on the model of the SUT.

Web application testing with MBT approach requires us to use a model. A model represents the correct behaviour of a system. The specification of the system, which is a documentation of what the system is capable of doing,

specifies what the system does when certain elements are used and what reaction the system should give on those used elements. With the help of a model, we can generate algorithmically test cases to verify if the SUT is behaving as designed or not. A model can be described in different ways. It may be noted that each model-based testing technique uses a different model to generate test cases.

Testing web application requires documents that specify what their correct behavior is. Lack of documentation is a critical problem in small web applications for testing. Test cases are commonly designed based on program source code. This makes test case generation difficult, especially for testing at cluster levels. Further, this approach proves to be inadequate in component-based software development, where the source code may not be available to the developers. It is, therefore, desirable to generate test cases automatically from the software design documents, rather than source code or code-based specifications, test case generation from design documents allows test cases to be available early in the software development life cycle, which makes test planning more efficient. Another advantage of design-based tests is to test the compliance of the implementation with the design documentation (Samuel et al. 2007). This is not the case for source-code based testing. Further, in design-based tests, the generated test data is independent of any particular implementation of the design. Dynamic web applications have many interactions between client-side machine and server-side machine, such that code coverage is difficult to capture these complex interaction for adequate testing. This paper uses model-based testing to detect faults in web applications. We have chosen the model based approach for testing of web applications because it fully automates the testing process and, adapts quicker to the changes. Also, this approach takes less time, and it is less error prone if the system is modeled correctly.

3 Basic concepts

This section presents some basic concepts related to web application testing, such as overview of web applications, Java Server Pages (JSP), dynamic page validation, test cases and test scenarios.

3.1 Web applications

A web application (Hall and Brown 2008) is a program that compiles on the server side and displays the result on the client browser. It is created in a browser-supported programming language (such as the combination of JSP, Servlet, JavaScript, HTML and CSS) and relies on a web browser to render the application. Web applications are

mainly two types: Static web applications and Dynamic web applications.

3.1.1 Static web application

A static web application consists of simple web pages that are generated by the server according to the client request. Static web page is always displayed on client side machine. Static web application displays the same information with same context for all clients. The static web application is stored in HTML format and is always available to the web server over HTTP. Every web application with .html extension is not always a static web application (Hall and Brown 2008).

3.1.2 Dynamic web application

The dynamic web application is a server-side application written in a scripting language. It is stored and compiled on the server side and the output is displayed on the client side browser. This type of application is developed in JSP, servlet, Javascript, PHP, Ruby, etc. Dynamic web application development is a very difficult task for a developer. It can be developed by the experienced developers. There are some advantages of developing dynamic websites (Hall and Brown 2008) which are given below:

- Improves functionality in website.
- Easy to update the website.
- New contents can be added in site which help to improve the execution process of search engine.
- Collaboration of different sites is very easy.

3.2 JSP

In this section, first we discuss the basics of JSP along with its advantages. Then, we discuss the JSP implicit objects. Next, we describe the various types of scripting elements. Then, we discuss JSP processing. JSP is a server-side script language to handle HTTP requests and generates dynamic contents on the client side. This script links with other components of server for sending HTTP response (Hall and Brown 2008). Any compiler does not compile the server side script language so that it can be error-prone. Nowadays, most of the web applications use JSP and servlet script languages for developing dynamic web pages. JSP is a web page scripting language that can generate dynamic content while Servlets are Java programs that are already compiled which also create dynamic web content. Java later released JSP as a more flexible scripting alternative to Java Servlets. JSP has some advantages than Servlet language which are given below:

- JSP can be compiled into Java Servlets.
- It's easier to code in JSP than in Java.
- Nowadays, JSP and Java Servlets are usually used in conjunction.
- Java and JSP can be combined with HTML to provide dynamic contents for Web pages.
- JSP provides custom library, called taglibs, using HTML-like tags.
- JSP separates the dynamic contents of a web page from its presentation.

Due to the above advantages of JSP, it is used in our approach for testing dynamic web applications. This paper presents, model based testing (MBT) approach for dynamic web application testing (DWAT). The proposed approach considers JSP for test case generation. In this approach, implicit class object tags (ICOT) and directives are used for generating the test scenarios and test cases.

3.2.1 JSP implicit objects

JSP implicit objects are explicitly declared inside each page. JSP container provides JSP implicit objects library. JSP implicit objects are also called predefined variables. There are mainly nine types of JSP implicit objects, as shown in Table 1. These objects are created by JSP Engine during translation phase (while translating JSP to Servlet). They are created inside service method, so we can directly use them within Scriptlet without initializing and declaring them. Below we explain all the JSP implicit class objects (Ricca and Tonella 2000, 2001; Liu 2004; Fujiwara et al. 2011; Kung et al. 2000).

Request The main purpose of request implicit object is to get the data on a JSP page which has been entered by user on the previous JSP page. While dealing with login and signup forms in JSP, often the user is prompted to fill in those details. This object is then used to get those entered details on another JSP page (action page) for validation and other purposes.

Response It is used for modifying or deleting with the response which is being sent to the client(browser) after processing the request.

Out This is used for writing content to the client (browser). It has several methods which can be used for properly formatting output message to the browser and for dealing with the buffer.

Session It is the most frequently used implicit object. This is used for storing the users data to make it available on other JSP pages till the user session is active.

Application This is used for getting application-wide initialization parameters and to maintain useful data across whole JSP application.

Table 1 JSP implicit object tags

SI	Object	Implicit object library	Description
1	request	javax.servlet.http.HttpServletRequest	This is the HttpServletRequest object associated with the request
2	response	javax.servlet.http.HttpServletResponse	This is the HttpServletResponse object associated with the response to the client
3	out	javax.servlet.jsp.JspWriter	This is the PrintWriter object used to send output to the client
4	session	javax.servlet.http.HttpSession	This is the HttpSession object associated with the request
5	application	javax.servlet.ServletContext	This is the ServletContext object associated with the application
6	config	javax.servlet.ServletConfig	This is the ServletConfig object associated with the page
7	pageContext	javax.servlet.jsp.PageContext	This encapsulates the use of server-specific features like higher performance JSPWriters
8	page	java.lang.Object	This is simply the synonym of <i>this</i> object, and is used to call the methods defined by the translated servlet class
9	exception	javax.servlet.jsp.JspException	The exception object allows the exception data to be accessed by the designated JSP

Config This is a Servlet configuration object and mainly used for accessing configuration information such as servlet context, servlet name, configuration parameters, etc.

pageContext It is used for accessing page, request, application and session attributes.

Page Page implicit object is a reference to the current Servlet instance (Converted Servlet, generated during translation phase from a JSP page). We are not covering it in detail as it is rarely used and not a useful implicit object while building a JSP application.

Exception Exception implicit object is used in exception handling for displaying the error messages. This object is only available to the JSP pages. This object has a tag named `isErrorPage` which is true.

We have also presented implicit class objects with examples in Fig. 1 and executable output of presented examples are given in Fig. 2.

3.2.2 Scripting elements

Scripting elements¹ are used for writing Java code inside the JSP page. There are three types of scripting elements:

1. *Scripting tag* Scripting tag is used to execute Java source code in JSP. The scripting elements of a Java Server Page are utilized to perform server-side operation in a JSP. JSP scripting elements are also called scriptlets and perform Java and Java script functionality (Hall and Brown 2008; Conallen 2002; Ricca and Tonella 2000).
2. *Expression tag* Expression tag is used for writing the output stream. The Expression element (Hall and Brown 2008; Conallen 2002; Ricca and Tonella 2000) contains a Java expression that returns a value. This

value is then written to the HTML page. The Expression tag can contain any expression that is valid according to the Java Language Specification. This includes variables, method calls than return values or any object that contains a `toString()` method.

The Java expression is evaluated, converted to a string, and inserted in the page. This evaluation is performed at run-time (when the page is requested), and thus has full access to information about the request.

Remember that XML elements, unlike HTML ones, are case sensitive. So be sure to use lowercase letters.

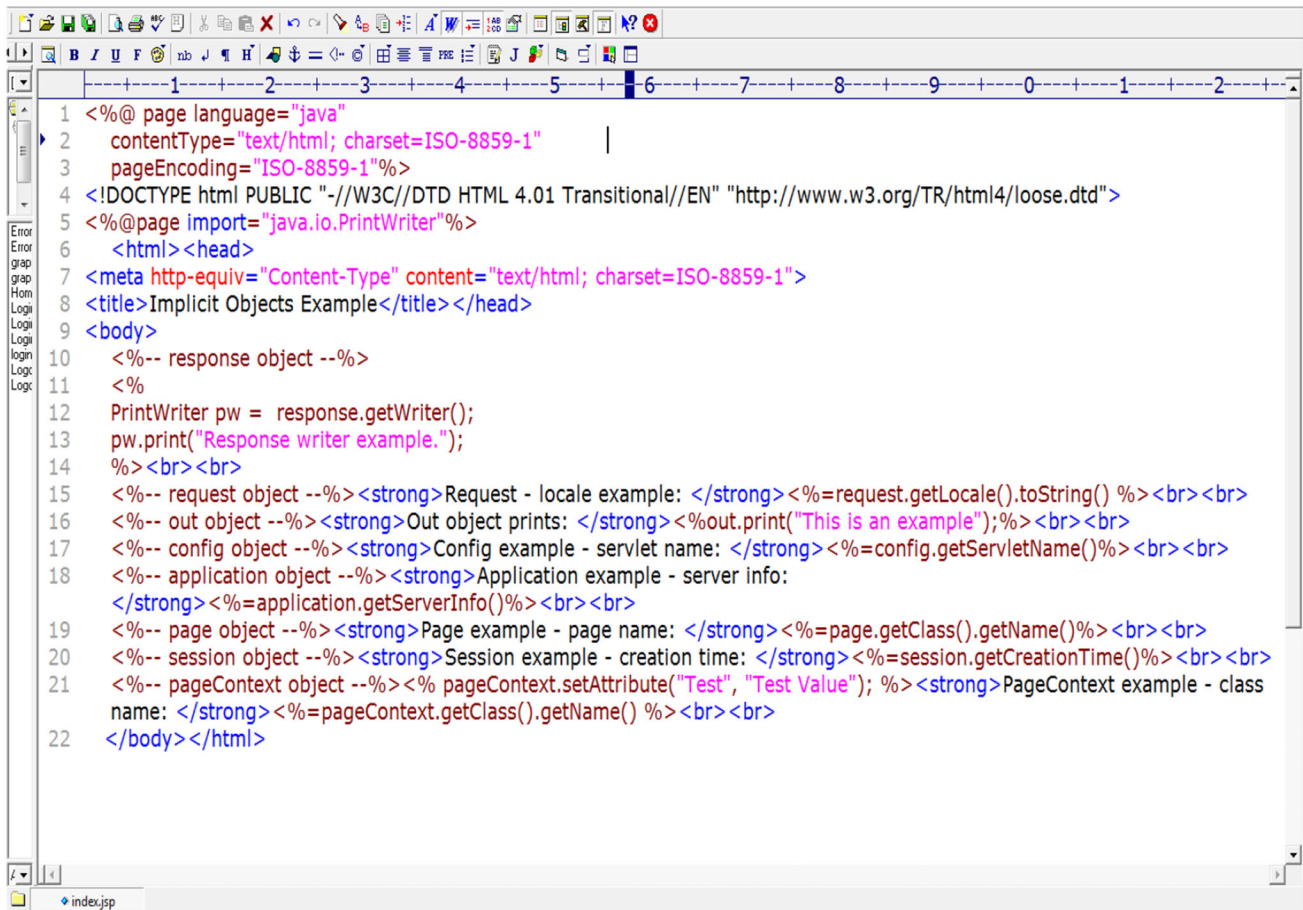
3. *Declaration tag* Declaration tag is used to define fields and methods inside the JSP code. This tag always writes outside the `service()` method of auto generated servlet. So, it doesn't have any memory to each request. A declaration can consist of either methods or variables. Static constants are a good example of what to put in a declaration.

The declaration always ends the Declaration object tag with a semicolon (the same rule as for a Scriptlet, but the opposite of an Expression) e.g. `<%int i = 0; %>`. You can use variables or methods that are declared in packages imported by the page directive, without declaring them in a declaration element. A declaration has translation unit scope, so it is valid in the JSP page and any of its static include files. A static include file becomes part of the source code of the JSP page and is any file included with an include directive or a static resource included with a `<jsp:include >` element. The scope of a declaration does not include dynamic resources included with `<jsp:include >`.

3.3 Dynamic page validation

The internal structure of dynamic web application is found in the JSP pages. Here, internal structure means

¹ <http://www.studytonight.com/jsp/jsp-scripting-element.php>.



```

1 <%@ page language="java"
2   contentType="text/html; charset=ISO-8859-1"
3   pageEncoding="ISO-8859-1"%>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
5 <%@page import="java.io.PrintWriter"%>
6 <html><head>
7 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8 <title>Implicit Objects Example</title></head>
9 <body>
10 <%-- response object --%>
11 <%
12   PrintWriter pw = response.getWriter();
13   pw.print("Response writer example.");
14   %><br><br>
15 <%-- request object --%><strong>Request - locale example: </strong><%=request.getLocale().toString() %><br><br>
16 <%-- out object --%><strong>Out object prints: </strong><%out.print("This is an example");%><br><br>
17 <%-- config object --%><strong>Config example - servlet name: </strong><%=config.getServletName()%><br><br>
18 <%-- application object --%><strong>Application example - server info:
19 </strong><%=application.getServerInfo()%><br><br>
20 <%-- page object --%><strong>Page example - page name: </strong><%=page.getClass().getName()%><br><br>
21 <%-- session object --%><strong>Session example - creation time: </strong><%=session.getCreationTime()%><br><br>
22 <%-- pageContext object --%><% pageContext.setAttribute("Test", "Test Value"); %><strong>PageContext example - class
    name: </strong><%=pageContext.getClass().getName() %><br><br>
    </body></html>

```

Fig. 1 Example of implicit class object tags in JSP page

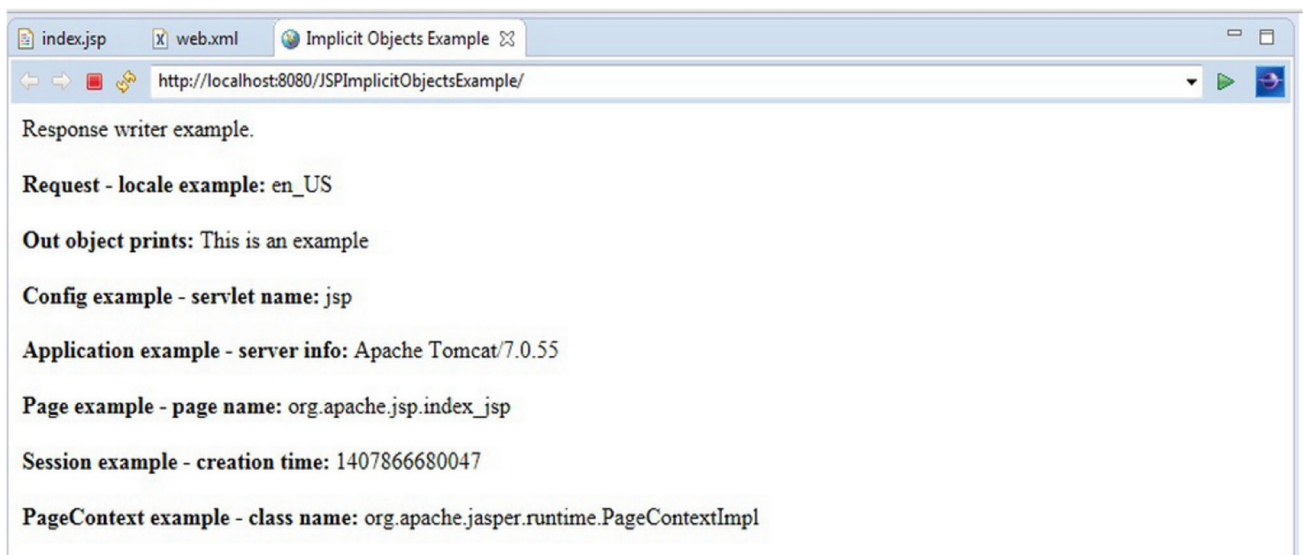


Fig. 2 Output of the example implicit class object tags given in Fig. 1

how the dynamic web applications are represented and with what elements. It may be noted that the dynamic web applications are represented in JSP through elements

such as implicit class objects, scripting elements, etc. This structure information is used for identification of test coverage criterion. Every test case of a dynamic page

depends on the test coverage criterion. Test cases for a web application are sequences of def-use (Definition-use) coverage criterion in the dynamic web page. The functional test cases for a web application depends on the visited path and the input values which are given to the forms. Some dynamic web testing approaches based on coverage criteria are given below:

1. Page testing: Every page in the site is visited at least once in some test case.
2. Hyperlink testing: Each hyperlink form within every page in the site is traversed at least once.
3. Definition-use testing: Each navigation from every definition of a variable to every use of it, forming a data dependence, is exercised at least once.
4. All-paths testing: Every path in the site is traversed by some test case at least once.

3.4 Test case and test scenario

A test case is the triplet [I, D, O], where I is the initial state of the system at which the test data is supplied as input, D is the test data which is supplied as input to the system and O is the expected output of the system (Booch and Jacobson 2005; Kanjilal and Bhattacharya 2004; Kung et al. 2000). Test cases are low level actions and they can be derived from test scenarios. The output produced by the execution of the software with a specific test case provides a specification of the actual software behavior (Booch and Jacobson 2005).

Test scenarios are sequence of test cases, which are to be executed. Test scenarios are test cases that ensure that all flows are tested from *start* to *end*. Before executing the test scenarios, the test cases for each scenario have to be developed. Test scenarios are the high level classification of test requirements grouped together depending on the functionality of a module and they can be derived from use cases. Test scenarios are prepared by reviewing the functional requirements, and preparing logical groups of functions that can be further broken into test procedures Booch and Jacobson (2005).

4 Generating test scenarios for dynamic web applications

This approach uses JSP script language for dynamic web testing. This approach generates test scenarios for testing the basic functionalities of web pages. The flow chart for test case generation for web applications is shown in

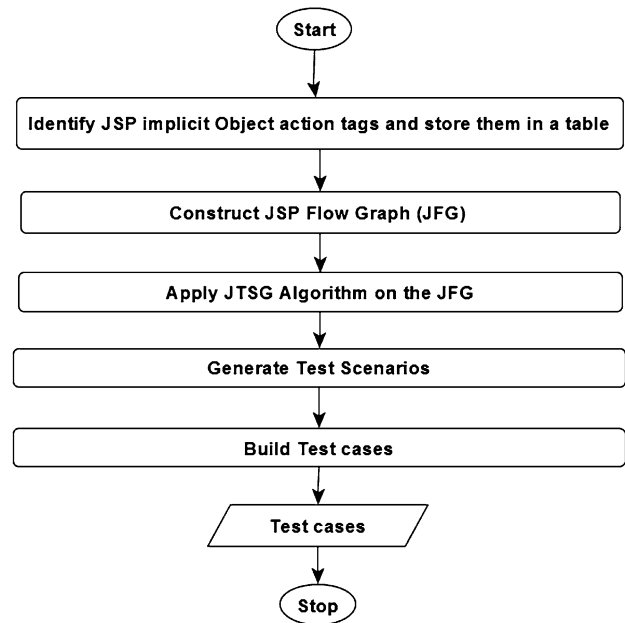


Fig. 3 Flowchart for test case generation for web applications

Fig. 3. The step-wise procedure for test scenario generation for a web application, is presented below.

Step 1. Identify the feasible requirements of a given web application First, the feasible functional requirements of the web application are identified.

Step 2. Develop the basic models for the requirements Then, models such as UML use case diagram, class diagram, etc. are developed according to the feasible requirements. These models are useful for covering the basic requirements of the project.

Step 3. Implement the web application according to the developed models and requirements, in JSP script language After developing the models for feasible requirements, the web application is implemented in JSP language according to the models.

Step 4. Identify JSP implicit object action tags and store them in a table JSP container provides the JSP implicit object tags library inside the JSP pages. The JSP implicit object tags are given in Table 1. The implicit object tags are identified and stored in a table. The implicit object tags for our case study are shown in Figs. 7 and 8.

Step 5. Construct JSP Flow Graph (JFG) After identifying the JSP implicit object tags, an intermediate graph called JSP Flow Graph (JFG) is constructed. For constructing the JFG, first the JSP code is instrumented using statement numbers. After that, the alias names are given. The alias names for our case study are shown in Table 2. According to the structure of JSP code, Alias

Table 2 Alias representation of JSP implicit class object tags for login use case

SI	Line no.	Alias name
1	1	A
2	2–18	B
3	3–6	B.1
4	7–12	B.2
5	8	B.2.1
6	9–16	B.2.2
7	10	B.2.2.1
8	11	B.2.2.2
9	logincheck.jsp	C
10	1	D
11	2–20	E
12	3–6	E.1
13	7–19	E.2
14	8–18	F
15	4, Home.jsp	G
16	17, error.jsp	H

table and flow of control in JSP page, and the JFG for JSP page are constructed.

Step 6. Generate test scenarios for the functionalities of web application from the JFG using JTSG algorithm The JTSG algorithm (given in Algorithm 1) is applied on JFG for generating the test scenarios.

Step 7. Build the test cases for the web application based on the generated test scenarios After generating the test scenarios, the test cases are developed semi-automatically. According to the test scenarios, every node of JFG is taken one by one and the instrumented code is extracted according to the alias table. There are many tags in the extracted code. By using the tags, test cases are developed.

In JTSG (Algorithm 1), JSP Flow Graph (JFG) is supplied as input. The outcome of JTSG algorithm are the test scenarios.

First, the array of *nodes* is initialized. Then, the array $TS[i]$ is created and initialized for storing test scenarios. After that, all the nodes of JFG are stored in $node[i]$. If the condition ($node[i] == End$) is true then, the *CN* (Current Node) is stored into $TS[i]$. If condition is false, then *left node of CN* is checked up to *end node*. If the condition of the while loop is true, then *left node of CN* is stored into $TS[i]$. After that, $TS[i]$ is incremented. When the condition of the while loop becomes false, then the right node of *CN* is stored into $TS[i]$ and $TS[i]$ is incremented. Finally, the test scenarios stored in $TS[i]$ are displayed. Then, the test cases are developed according to the generated test scenarios.

Algorithm 1 JTSG Algorithm

Input JFG: JSP Flow Graph

Output TS[]: Test Scenarios

```

1: Initialize node[]= $\phi$ 
2: Initialize TS[i]= $\phi$ 
3: if node[i] == End then
4:   TS[i]=node[i]; //TS[i]= Test scenarios
5: else
6:   while (CN  $\rightarrow$  left  $\neq$  End) do //CN= Current node in JFG
7:     TS[i]=CN  $\rightarrow$  left
8:     TS[i]=TS[i+1];
9:   end while
10:  TS[i]=CN  $\rightarrow$  right
11:  TS[i]=TS[i+1];
12:  Display "Test Sequences  $\rightarrow$  TS[i]";
13: end if

```

5 Case study

Our proposed approach can be used by software testers and web developers in software industries to test the web applications written in any web development language such as JSP, servlets, Java script, HTML etc. This section considers a Login page for illustrating and implementing the proposed approach. The present approach considers the login process in five web pages. These are as follows: login.jsp and loginCheck.jsp in Fig. 4, home.jsp and error.jsp as shown in Fig. 5 and logout.jsp as shown in Fig. 6. The proposed approach uses JSP directives “contentType” and “pageEncoding” for defining the type of content and the encoding scheme respectively in Login page. After that, design a form with two fields *username* and *password*. The *username* and *password* attributes are for retrieving input values from other fields in another page. The attribute values of *username* and *password* are “vikas” and “panthi”. When credentials get verified, then the form is submitted to loginCheck.jsp page respectively which is shown in Fig. 4b.

This section shows how the compilation process of loginCheck page takes place. Validation page defines the basic page directive and initializes “request.getParameter” field for obtaining the “username” and “password” values. Then, the username and password and obtained as “request.getParameter(“username”)” and “request.getParameter(“password”)” respectively. After obtaining the values and matching with the correct credentials, which are “vikas(useranme)” and “panthi(password)”. The page transfers the control to the home page, otherwise control is transferred to the error page. This case study can also consider user values with database values. But for simplicity, database concepts are not considered here. After that, define session variables for storing the temporary values. This session can store credential values till the session expires. In this page, other pages can be called

```

1 <html>
2 <head>
3 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4 <title> JSP Page </title>
5 </head>
6 <body>
7 <h1>Login Page</h1>
8 <center>
9 <h2>Signup Details</h2>
10 <form action="LoginCheck.jsp method="post">
11 <br/>Username: <input type="text" name="username">
12 <br/>Password: <input type="password" name="password">
13 <br/><input type="submit" value="Submit"
14 </form>
15 </center>
16 </body>
17 </html>

```

(a)

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 <title>JSP Page</title>
6 </head>
7 <body>
8 <%
9 String username=request.getParameter("username");
10 String password=request.getParameter("password");
11 if((username.equals("vikas") && password.equals("panthi")))
12 {
13     session.setAttribute("username", username);
14     response.sendRedirect("Home.jsp");
15 }
16 else
17     response.sendRedirect("Error.jsp")
18 %>
19 </body>
20 </html>

```

(b)

Fig. 4 a JSP page of Login.jsp b JSP page of LoginCheck.jsp

```

1 <%@page contentType="text/html" pageEncoding="UTF-8" errorpage="Error.jsp"%>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 <title>JSP Page</title>
6 </head>
7 <body>
8 <br/><br/><br/><br/><br/>
9 <center>
10 <h2>
11 <% String a=session.getAttribute("username").toString();
12 out.println("Hello" +a);
13 %>
14 </h2>
15 <br/>
16 <br/><br/><br/><br/><br/>
17 <a href="logout.jsp">Logout</a>
18 </center>
19 </body>
20 </html>

```

(a)

```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html;
5 charset=UTF-8">
6 <title>JSP Page</title>
7 </head>
8 <h1>Some Error has occurred, Please try again later... </h1>
9 </body>
10 </html>

```

(b)

Fig. 5 a JSP page of Home.jsp b JSP page of Error.jsp

using “response.sendRedirect()” method. If credential values are matched with original values, then the home page is called otherwise, the error page is called. Now, let us describe the home page. Here, we have taken a new directive attribute called errorpage, in which, if there is any error in home page, then the error page is called. This directive is mainly used for error handling. In the home page, session attribute is used to extract and display the username on the client browser. Logout button is added in the home page for logging out purpose. In logout page, the session “session.removeAttribute” is used to remove both of the attributes. This session is mainly used when control comes to this page, then the session will expire and the control is transformed to the home page to display error message.

The implementation of the proposed approach is explained below taking Login use case as the case study. First, instrument all the JSP source codes of Login use case. Then, extract the implicit class object of every JSP page of Login use case. All the implicit object tables are shown in Figs. 7, 8 and 9. The alias representation of JSP implicit class object is given in Table 2.

Then, construct the JFG (JSP Flow Graph). JFG of Login use case is shown in Fig. 10. JFG is a combination of control flow graph and data flow graph. Then, apply the proposed algorithm JTSG, for generating test scenarios using JFG. JFG is supplied as an input in link list form to the JTSG algorithm. The proposed algorithm is implemented in JAVA for generating test scenarios. The


```

1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 <title>JSP Page</title>
6 </head>
7 <body>
8 <%
9 session.removeAttribute("username");
10 session.removeAttribute("password");
11 session.invalidate();
12 %>
13 <h1>Logout was done successfully.</h1>
14 </body>
15 </html>
    
```

Fig. 6 JSP page of Logout.jsp

generated test scenarios are shown in Fig. 11. Finally, the test cases are generated from the test scenarios. The generated test cases are shown in Table 3.

6 Comparison with related works

This section provides some existing approaches to dynamic web application testing (DWAT).

Alshahwan and Harman (2011) proposed a web application testing tool named search based web application tester (SWAT) with dynamic and static seeding. They proposed a search based testing algorithm. They considered PHP web application for implementing their approach. In their paper, they have discussed many issues raised by web

applications such as dynamic type binding, user interface inference, etc.

Törsel (2011) presented a model-based testing approach for user interface level testing. They developed a prototype for generate test oracles from model information, and they transformed abstract test cases to executable test scripts.

Li et al. (2008) presented a practical test model and test approach for web applications based on use cases and their corresponding sequence diagrams. They proposed a hierarchical profile use case model called use case transition model (UCTM). They traversed the UCTM from top to bottom and converted it into restricted message on vertex graph (RMOVG). Every vertex in RMOVG represents one message in the sequence diagram. They proposed constraint message coverage (CMC) criterion for test case generation.

Dai and Chen (2008) presented a technique for automatic test case generation. The created test suite not only covers the specification but also ensures that fault-sensitive execution sequences are exercised. Their approach considered multi-tier web applications for testing.

Boni et al. (2009) introduced the architecture of a system which tries to fully automate the test case generation process for web applications based on agile framework named automatic testing platform (ATP). Their tool covered many testing aspects, such as unit testing, system testing, test case execution and reporting. Their tool is based on the usage of different pluggable testing tools like JUnit, TestNG and Selenium.

Hajiabadi and Kahani (2011) proposed a model based technique to test web applications from their structural models. They applied several ontologies and mapping tools, test cases for filling forms for automatically

Form Name		Login.jsp	
PageContentType="text/html"		PageEncoding="UTF-8"	
head	meta	Httpequiv="ContentType" Content="text/html, charset=UTF-8"	
	title	JSP Page	
body	H1	Login page	
	H2	Signup details	
	form	Action="post"	Method= Logincheck.jsp
	input	Type="text"	Name= username
		Type="password"	Nmae= password
	-----	-----	-----

(a)

Form Name		Logincheck.jsp	
PageContentType="text/html"		PageEncoding="UTF-8"	
head	meta	Httpequiv="ContentType"	Content="text/html, charset=UTF-8"
	title	JSP Page	
body		<% String usermae=request.getParameter("username"); String password=request.getParameter("password"); if((username.equals("vikas")&&password.equals("panthi"))) { session.setAttribute("username", username); response.sendRedirect("Home.jsp"); } else response.sendRedirect("Error.jsp") %>	
		-----	-----

(b)

Fig. 7 a Implicit object table of Login.jsp b Implicit object table of loginCheck.jsp

Form Name		Home.jsp	
PageContentType="text/html"		PageEncoding="UTF-8"	
head	meta	Httpequiv="ContentType"	Content="text/html, charset=UTF-8"
	title	JSP Page	
body	br	5	
	H2	<%String a=session.getAttribute("username"). toString(); out.println("Hello" +a); %>	
	Br	5	

(a)

Form Name		Logout.jsp	
PageContentType="text/html"		PageEncoding="UTF-8"	
head	meta	Httpequiv="ContentType"	Content="text/html, charset=UTF-8"
	title	JSP Page	
body	H1	Logout was done successfully..	

(b)

Fig. 8 a Implicit object table of home.jsp b Implicit object table of logout.jsp

Form Name		Error.jsp	
PageContentType="text/html"		PageEncoding="UTF-8"	
head	meta	Httpequiv="ContentType"	Content="text/html, charset=UTF-8"
	title	JSP Page	
body	H1	Some Error has occurred, Please try again later...	

Fig. 9 Implicit object table of error.jsp

generated models and evaluated dynamic features of the web applications. Their approach was implemented as MBTester tool and applied to a few web applications.

Monsma (2015) proposed a model based testing tool called GvST for web application testing. Their developed tool was mainly used for GUI testing. They generated test cases based on the model of the web application. They considered phantomJ, a headless browser, to access the web application. After that, they established connection between GvST tool and headless browser. Finally, they generated test cases based on GUI. Achkar (2010) compared many tools of web application testing such as NModel, conformiq test generator tool, TestOptimal etc. They compared the results of DDI health case study in their research study.

In this paper, we have proposed a novel approach for test case generation for dynamic web application (DWA). In

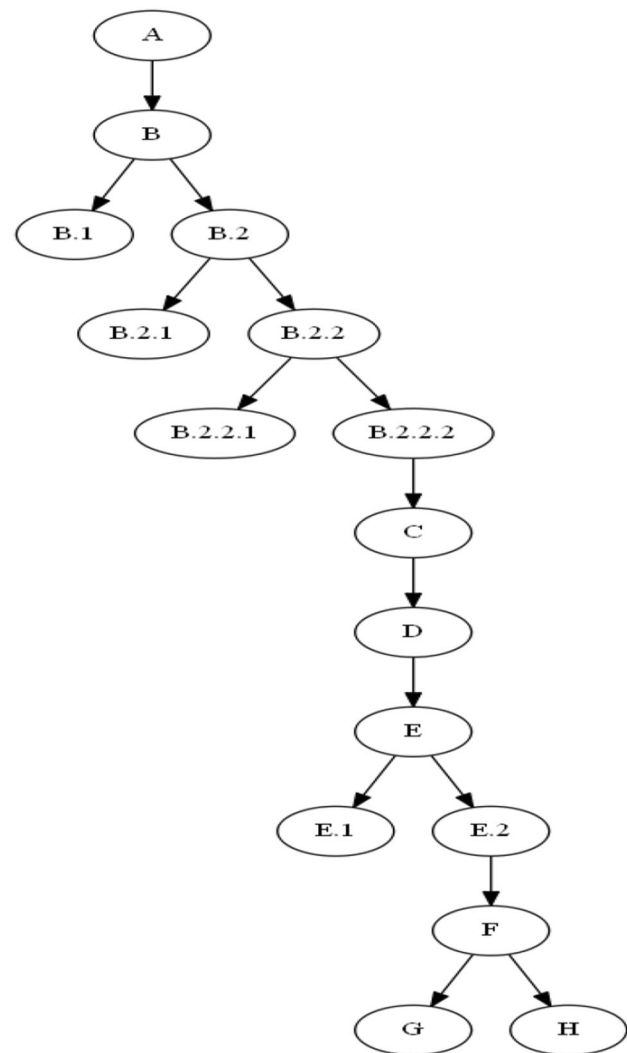


Fig. 10 JFG for login use case

Fig. 11 Generated test scenarios for login use case

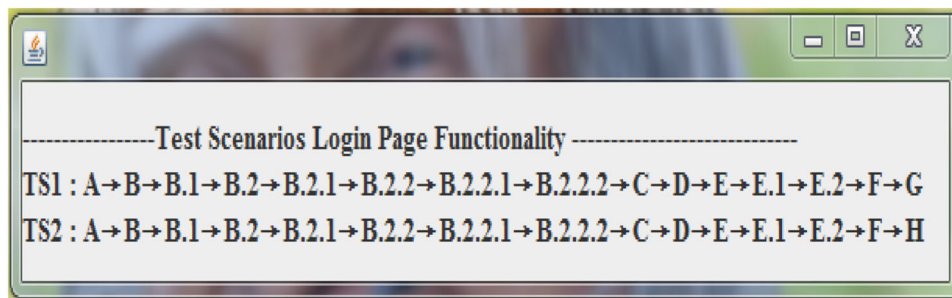


Table 3 Test cases for login use case

JSP page	Tags	Test data	Expected result	Actual result
Login.jsp	<title >		JSP Page	JSP Page
	<h1 >		Login Page	Login Page
	<h2 >	Sign up details	Sign up details	Sign up details
	<form >	Username, Password	Username, Password, Action:post, loginCheck.jsp	Username, Password, Action:post, loginCheck.jsp
loginCheck.jsp	<title >		JSP Page	JSP Page
	<%...% >	username = vikas, password = panthi	home.jsp,	home.jsp
loginCheck.jsp	<title >		JSP Page	JSP Page
	<%...% >	username = anil, password = mishra	error.jsp,	error.jsp
home.jsp	<title >		JSP Page	JSP Page
	<%...% >	username = vikas, password = panthi	Hello vikas	Hello vikas
	<a...href/a >		href = logout.jsp	href = logout.jsp
home.jsp	<title >		JSP Page	JSP Page
	<%...% >	username = anil, password = verma	href = error.jsp	href = error.jsp
	<a...href/a >		href = error.jsp	href = error.jsp
Error.jsp	<title >		JSP Page	JSP Page
	<h1 >	username = anil, password = mishra	Some Error has occurred, Please try again later	Some Error has occurred, Please try again later
Logout.jsp	<%...% >	Clickbutton	Referesh username, password	Referesh username, password
	<h1 >		Login Page	Login Page
	<h1 >		logout was done successfully	logout was done successfully

which, we have considered JSP pages for dynamically tracing the activity of web application. We have proposed testable graph called JSP Flow Graph (JFG). In JSP page, we fetched all Implicit Class Object for converting JSP into JFG. Finally, we have generate the test scenarios using JFG.

In Table 4, we have shown the achieved Test coverage for different case studies. In this table, we have calculated the following parameters for 5 (five) case studies. TN (total nodes), TT (total transitions), CN (covered nodes), CT (covered transitions), NC% (node covered percentage), TC% (transitions covered percentage), TL (total lines of code), TLC (total lines of code covered), TLC% (total lines of code covered percentage). The case studies that, we have considered are: login case study, student registration, on-line ticket reservation system, book reservation system, on-line book shop. All the case studies are developed by UG

(Undergraduate B.Tech)/PG (Postgraduate M.Tech) students of Department of Computer Science and Engineering at National Institute of Technology, Rourkela, Odisha.

In Table 4, we find that node and transition coverages are directly proportional to test cases. LOC coverage is directly proportional to node and transition coverages. If we have covered all nodes and transitions, so indirectly we have covered source code. So, this techniques improves the quality of web application.

7 Conclusion

In this paper, an approach is presented for testing the functional requirements of web applications. JSP script language is used in this approach for dynamic web

Table 4 Achieved test coverage for different case studies

	TC_{ID}	TN	TT	CN	CT	NC%	TC%	TL	TLC	TLC%
Login case study	TS_1	16	15	15	14	93.75	93.33	82	69	84.14
	TS_2	16	15	15	14	93.75	93.33	82	44	53.63
Student registration	TS_1	23	27	18	16	78.26	59.25	163	139	85.27
	TS_2	23	27	13	16	56.52	59.25	163	119	73.00
	TS_3	23	27	20	21	86.95	77.77	163	147	90.18
	TS_4	23	27	8	9	34.78	33.33	163	72	44.17
	TS_5	23	27	13	14	56.52	51.85	163	112	68.71
On-line ticket reservation system	TS_1	37	35	34	31	91.89	88.57	206	187	90.77
	TS_2	37	35	29	27	78.37	77.14	206	162	78.64
	TS_3	37	35	27	31	72.97	88.57	206	169	82.03
	TS_4	37	35	30	32	81.08	86.48	206	172	83.49
	TS_5	37	35	27	29	72.97	82.85	206	159	77.18
	TS_6	37	35	31	30	83.78	85.71	206	174	84.46
	TS_7	37	35	15	17	40.54	48.57	206	97	47.08
	TS_8	37	35	17	19	45.94	54.28	206	109	52.91
	TS_9	37	35	33	32	89.18	91.42	206	191	92.71
Book reservation system	TS_1	31	33	29	31	93.54	93.93	189	178	94.17
	TS_2	31	33	19	21	61.29	63.63	189	119	62.96
	TS_3	31	33	27	25	87.09	75.75	189	156	82.53
	TS_4	31	33	23	25	74.19	75.75	189	137	72.48
	TS_5	31	33	15	17	48.38	51.51	189	112	59.25
	TS_6	31	33	19	21	61.29	63.63	189	131	69.31
On-line book shop	TS_1	44	42	38	41	86.36	97.61	347	287	82.70
	TS_2	44	42	34	33	77.27	78.57	347	254	73.19
	TS_3	44	42	41	40	93.18	95.23	347	304	87.60
	TS_4	44	42	26	25	59.09	59.52	347	227	65.41
	TS_5	44	42	42	41	95.45	97.61	347	318	91.64
	TS_6	44	42	39	40	88.63	95.23	347	296	85.30
	TS_7	44	42	41	39	93.18	92.85	347	302	87.03
	TS_8	44	42	29	32	65.90	76.19	347	263	75.79
	TS_9	44	42	27	29	61.36	69.04	347	241	69.45
	TS_{10}	44	42	39	41	88.63	97.61	347	316	91.06
	TS_{11}	44	42	21	23	47.72	54.76	347	211	60.80
	TS_{12}	44	42	40	38	90.90	90.47	347	289	83.28
	TS_{13}	44	42	35	37	79.54	88.09	347	282	81.26

TN Total nodes, TT total transitions, CN covered nodes, CT covered transitions, $NC\%$ node covered percentage, $TC\%$ transitions covered percentage, TL total lines of code, TLC total lines of code covered, $TLC\%$ total lines of code covered percentage

application testing. In this paper, Login use case is considered as the case study. First, the implicit class object tags are identified, and the JFG is constructed for the Login JSP page. Then, the functionalities in JFG are identified for test scenario generation. Then, JTSG algorithm is applied on the JFG for generating the test scenarios of the web page. Finally, test cases are developed for the web page. In future, we will generate the test scenarios based on service-oriented architecture (SOA).

There are many optimization and prioritization techniques exist for optimization such as Ant Colony Optimization, Fire Fly Optimization, Cuttlefish Optimization etc. We will try to apply these techniques in future.

Acknowledgements This work is supported by a Grant No. SR/FST/ETI-359/2014(C) for the FIST-2014, from Department of Science and Technology (DST), Government of India. Carried out by Department of Computer Science & Engineering, National Institute of Technology, Rourkela, Odisha, India.

References

- Achkar H (2010) Model based testing of web applications. STANZ-2010, Sydney, Australia, 26-27 August 2010.
- Afzal W, Torkar R, Feldt R (2008) A systematic mapping study on non-functional search-based software testing. In: International conference on software engineering and knowledge engineering, SEKE. p 488493
- Afzal W, Torkar R, Feldt R (2009) A systematic review of search-based testing for non-functional system properties. *Inf Softw Technol Elsevier* 51(6):957–976
- Ali S, Briand LC, Hemmati H, Panesar-Walawege RK (2010) A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Trans Softw Eng* 36(6):742–762
- Alshahwan N, Harman M (2011) Automated web application testing using search based software engineering. In: 26th international conference on automated software engineering, IEEE Computer Society, pp 3–12
- Banerjee I, Nguyen B, Garousi V, Memon A (2013) Graphical user interface (gui) testing: systematic mapping and repository. *Inf Softw Technol* 55(10):1679–1694
- Barmi ZA, Ebrahimi AH, Feldt R (2011) Alignment of requirements specification and testing: a systematic mapping study. In: IEEE fourth international conference on software testing, verification and validation workshops (ICSTW), pp 476–485
- Boni G, Juan CD, Hugo APG (2009) Automatic functional and structural test case generation for web applications based on agile frameworks. In: IEEE 5th international workshop on automated specification and verification of web systems, pp 1–15
- Booch G, Jacobson I (2005) The unified modeling language user guide, 3rd edn. Pearson Education India, Noida
- Conallen J (2002) Building web applications with UML, 2nd edn. Addison-Wesley Publishing Company, Boston
- Dai Z, Chen M-H (2008) Automatic test case generation for multi-tier web applications. In: 9th IEEE international workshop on web site evolution (WSE 2007), pp 39–43
- Engström E, Runeson P (2011) Software product line testing—a systematic mapping study. *Inf Softw Technol* 53(1):2–13
- Engström E, Runeson P, Skoglund M (2010) A systematic review on regression test selection techniques. *Inf Softw Technol* 52(1):14–30
- Fujiwara S, Munakata K, Maeda Y, Katayama A, Uehara T (2011) Test data generation for web application using a uml class diagram with ocl constraints. *Innov Syst Softw Eng Springer* 7(4):275–282
- Grindal M, Offutt J, Andler SF (2005) Combination testing strategies: a survey. *Softw Test Verif Reliab* 15(3):167–199
- Hajjabadi H, Kahani M (2011) An automated model based approach to test web application using ontology. In: IEEE conference on open systems (ICOS), pp 348–353
- Hall M, Brown L (2008) Core servlets and JavaServer pages, 2nd edn. Pearson Education, Noida
- Jia Y, Harman M (2011) An analysis and survey of the development of mutation testing. *IEEE Trans Softw Eng* 37(5):649–678
- Kanjilal A, Bhattacharya S (2004) Static analysis of object oriented systems using extended control flow graph. In: 10th IEEE region conference TENCON, IEEE Computer Society, pp 310–313
- Kung DC, Liu C-H, Hsia P (2000) An object-oriented web test model for testing web applications. In: First Asia-Pacific conference on quality software, IEEE Computer Society, pp 111–120
- Li L, Miao H, Qian Z (2008) A uml-based approach to testing web applications. In: International symposium on computer science and computational technology (ISCCT'08), IEEE Computer Society, pp 397–401
- Liu C-H (2004) Data flow analysis and testing of java server pages. In: 28th Annual international computer software and applications conference, IEEE Computer Society, pp 114–119
- Memon AM, Nguyen BN (2010) Advances in automated model-based system testing of software applications with a gui front-end. *Adv Comput* 80(5):121–162
- Monsma JR (2015). Model based testing of web applications (Mater Thesis Computer Science). Radboud University, Nijmegen, Netherlands
- Neto PADMS, do Carmo Machado I, McGregor JD, De Almeida ES, de Lemos Meira SR (2011) A systematic mapping study of software product lines testing. *Inf Softw Technol* 53(5):407–423
- Ricca F, Tonella P (2000) Web site analysis: structure and evolution. In: IEEE international conference on software maintenance, ICSM2000, pp 76–86
- Ricca F, Tonella P (2001) Analysis and testing of web applications. In: 23rd international conference on software engineering, IEEE Computer Society, pp 25–34
- Samuel P, Mall R, Kanth P (2007) Automatic test case generation from UML communication diagrams. *Inf Softw Technol* 49(2):158–171
- Törsel A-M (2011) Automated test case generation for web applications from a domain specific model. In: 35th annual computer software and applications conference workshops (COMP-SACW), IEEE Computer Society, pp 137–142