CHALLENGES IN TESTING OF WEB APPLICATIONS

Miodrag Živković[1]*,
Tamara Živković[2]

[1]Singidunum University,
Belgrade, Serbia
[2]Faculty of Electrical Engineering,
University of Belgrade, Serbia

Abstract:
We have witnessed a huge expansion of web applications in the last decade. Traditional desktop applications are slowly losing the pace, as more and more software solutions are implemented as web applications, with the backend server and the client access through web browser on their machine. Web applications today are offering a wide spectrum of public, business and entertainment services, and they are designed to be used by human users, which implies certain level of quality and user interaction. Although web applications are efficient and convenient in most cases, their testing is still sometimes neglected and performed in inappropriate way. This paper will present challenges in web applications testing and review testing techniques and approaches which are considered to be state of the art, with focus on the techniques observable on the frontend.

Keywords:
web application testing, software testing, black box, front-end, html.

1. INTRODUCTION

Software testing in general can be summed as the set of activities performed with the goal of finding errors in software, ensuring that the software is behaving as defined in specification. On the other hand, software testing is also used to build the confidence in the correct behavior of the software under test, as it can show absence of major bugs and correct functionality of the software. The actual absence of bugs does not mean that there is no bugs in the software, as for any nontrivial software it is not possible to do exhaustive testing. It just shows that the current test set did not find any bugs. Keeping that in mind, it is of the highest importance that the test set covers the most important use case scenarios – the actual scenarios how the actual users would use the system after the deployment. Software testing can be divided into two major groups of techniques – white box testing and black box testing. Both of those techniques can be applied to all types of applications, including mobile applications, traditional desktop applications, and of course web applications.

White box testing techniques focus on how software is implemented, with the access to the code and internal mechanisms of the software components under test. These methods are efficient way for validation of software design and implementation.

Correspondence:
Miodrag Živković

e-mail:
mzivkovic@singidunum.ac.rs

DOI: 10.15308/Sinteza-2018-91-96

Black box testing techniques, on the other hand, do not assume access to the code, and testing is based on the actual requirements specification. Focus is on what software does, more specifically that the functionalities of implemented software are as specified in the documentation. The actual software under test is observed as a black box, and its functionality is tested by giving different input values and checking whether the output of the system matches the expected output defined in the requirement specification.

Web applications, together with the mobile applications, are the fastest growing software categories today. They support a large number of activities, and some of them are critical. For example, banking applications or medical applications with diagnostics based on the expert systems are considered critical, as any serious bug could cause financial loss, or even worse, loss of lives. Such applications must meet high standards of software quality. Web applications also bring a set of testing challenges on their own, as there is a large number of different technologies used for backend / frontend development. Frontend is usually based on set of technologies such as HTML, CSS and JavaScript, which are needed to create responsive and interactive web pages. They are usually called DHTML (Dynamic HTML), which is the combination of HTML, JavaScript, and Cascading Style Sheets (CSS) that will allow user page to change at runtime and reflect the state of the backend server. On the other hand, backend can be implemented in large variety of ways, for example, PHP together with a MS SQL database, or ASP .NET C# application, or even Java application in form of JSP. As the result, all parts of web application must be tested.

Although testing of web application has a lot in common with traditional desktop applications, there are some differences which come from some unique requirements of web applications. Web applications require testing techniques which will be as flexible as possible to handle dynamic nature of web applications, automatic (in best case scenario, not always possible), and take into account specific problems such as performance of the application under the high load (large number of concurrent users). Also, it is often required to verify the behavior of the web application on different web browsers and different operating systems. Testing of web applications is standardized by ISTQB authority [1] In this paper we will focus on the issues visible from the client side.

2. WEB APPLICATION TESTING CHALLENGES

Web application testing must use large number of new technologies, to cover all the components of the web application. Web applications can be susceptible to errors because of asynchronous communication, untyped JavaScript, event handling, timing/latency, browser dependence and much more. Abilities to observe and to control the behavior of application are two crucial aspects of testing. Observability in this aspect is referred as the difficulty of actually seeing the results of the tests. Unfortunately, general observability of web applications is low. When test set is run, sometimes individual test results can be more or less visible to the software tester who is performing testing. If all results are visible to the user on the screen, we say in that case that observability is high. This is not generally the case for web applications. Also, web applications generally have low controllability, due to multiple server side components which depend on underlying platform, used frameworks and programming languages. As the result, user interface produced by the backend server, which is usually HTML, can be generated dynamically in multiple different ways. In this paper, we will focus more on the results visible on the screen on the client side.

Just as a short example, we can observe the behavior of the page components in different browsers, especially if the page is written in HTML 5. HTML 5 is the latest version of HTML standard, and although it is available for several years now, it is known to still have some limitations with different browsers. Most of the issues are associated with Microsoft Internet Explorer or Edge, but there are some issues with older versions of other browsers as well. For example, let us consider standard HTML 5 input form element calendar. On large number of web applications, it is required to obtain information about the user's date of birth, which can be implemented easily with HTML 5 in the following way:

```
<form action="/action_page.php" method="post">
Enter your birthday: <input type="date" name="bd1">
</form>
```

This element is displayed in appropriate way in Google Chrome or Mozilla Firefox, and also in Microsoft Edge (Fig. 1), however, even the latest version of Internet Explorer has problems displaying it (Fig. 2). The same element is just displayed as simple input text field in Internet Explorer version 11.

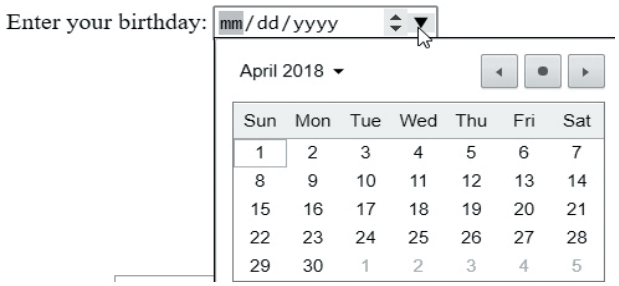

Fig. 1. HTML 5 calendar element in Google Chrome

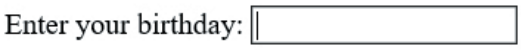

Fig. 2. HTML 5 calendar element in Internet Explorer version 11

In this case, if we wanted the same behavior of the page on all browsers, we would need to implement it in another way. Even worse, CSS and JavaScript also do not behave exactly the same on different browsers. In general, most of the common cross browser HMTL and CSS issues are caused by modern features and layout issues, which some browsers are not capable to show in proper way. This is actually a very common problem, especially if we need to support some older browser versions as well (again, most issues are with older Internet Explorer versions). Generally speaking, most of the core HTML and CSS functionalities, such as basic HTML elements, basic CSS colors and basic text styling would work over most of the browsers currently available. Problems generally arise when we want to use some of the newer HTML 5 and CSS 3 features, such as Flexbox, video/audio support or CSS grids. Even though some complex modern HTML elements such as <canvas>, <video> or <audio> have natural mechanisms for fallbacks, this does not necessarily mean that those mechanisms are actually implemented by the developer [2]. Therefore, it is a necessity to do thorough testing of all supported browsers, with major focus on Internet Explorer, to verify that all page components are displayed in an appropriate way, and that any needed fallback is actually implemented. In fact, fallback in case of <video> element would look like on Fig. 3.

```
<video id="video" controls preload="metadata"
    poster="img/poster.jpg">
    <source src="video/demo_clip.mp4" type="video/mp4">
    <source src="video/demo_clip.webm" type="video/webm">
    <source src="video/demo_clip.ogg" type="video/ogg">
    <!-- Flash fallback -->
    <object type="application/x-shockwave-flash"
        data="flash-player.swf?videoUrl=video/demo_clip.mp4"
        width="1024" height="576">
    <param name="movie"
        value="flash-player.swf?videoUrl=video/demo_clip.mp4" />
    <param name="allowfullscreen" value="true" />
    <param name="wmode" value="transparent" />
    <param name="flashvars" value="controlbar=over&amp;
        image=img/poster.jpg&amp;
        file=flash-player.swf?videoUrl=video/demo_clip.mp4" />
    <img alt="demo_clip image"
        src="img/poster.jpg" width="1024" height="428"
    title="No video playback possible, please download the video" />
    </object>
    <!-- Offer download -->
    <a href="video/demo_clip.mp4">Download MP4</a>
</video>
```

Fig. 3. <video> fallback in case it is not supported by browser

Fallback content is usually added in between opening and closing tags, and browser which does not support this feature will effectively ignore the element on the outside, and actually run the nested content. In this example, for older Internet Explorer versions, video fallback to Flash is offered, and even if the Flash player does not work, simple download link is offered so user can at least still open and view the content. CSS is even better at callbacks than HTML, as if browser runs at rule it does not understand, it will just skip it completely without applying it. Since Internet Explorer is generally the biggest problem, it is possible to add conditional comments inside HTML syntax, which can then be used to selectively apply rules to different versions of Internet Explorer, and the actual syntax is shown in Fig. 4.

```
<!--[if lte IE 8]>
<link rel="stylesheet" type="text/css" href="ie-css.css" />
<![endif]-->
```

Fig. 4. Conditional comments for handling different versions of IE

This actual block of code will apply specific CSS rules in case the browser is Internet Explorer 8 or older (version less than or equal to IE 8).

Security and robustness of web application is also of the biggest importance, so implementation of input validation is a must. It is referring to checking if the user input inside forms is valid and safe. For example, malicious user could try to insert SQL injection inside form, targeting the database on the backend server. SQL injection is targeting vulnerabilities in user input validation,

such as bad filtering of the escape character, and causes unexpected code execution. If we consider the statement which is extracting record with the given username from the table of users:

statement = "SELECT * FROM users WHERE name = '" + userName + "';"

If the userName is entered in the input form on the client side in a specific way, malicious user can try to execute SQL code in a way author of the code did not want. In case userName variable is set to value ' or '1'='1, the following query will come to the database:

SELECT * FROM users WHERE name = '' OR '1'='1';

Since subexpression '1'='1' is always true, this query would actually select all valid usernames from the database. This is why validation of the user input is necessary, and must be done on JavaScript level as the input sanitization, together with the additional check on the PHP level for example, before triggering actual query towards the database.

3. TESTING TECHNIQUES

As discussed in the introduction, both white box and black box testing techniques can be used when testing web applications. Some of the traditional techniques are not suitable for AJAX, and for other technologies there are also some limitations [3]. We will, however, try to summarize most frequently used techniques.

White box testing

In traditional software testing, white box testing is focusing on the actual code and internal structure of the software system under test. Similarly, when applied to web applications, we assume knowledge of the internal structure of the system, and usually structural models are used. However, it must be noted here that due to highly dynamic structure of modern web applications, and presence of several different technologies, white box techniques are just partially applicable to web application as a whole. Of course, each individual component can be tested on unit or integration level with traditional white box code coverage techniques. One of the available structural models is the Navigation model, where graph is used to represent the web site, with each page being the node, and every link being the edge. However, this model does not cover asynchronous behavior and dynamic changes of the modern web applications. With this model we cannot observe dynamic changes and the states of the HTML page which can happen during the execution.

Another approach is to use traditional white box code coverage techniques, including control flow testing and data flow testing, adapted for web applications. Control flow testing is based on creating control flow graph. Nodes of this graph are statements of the code executed on the web server, while the edges represent transfer of control between nodes. This is usually code and statements from different technologies, such as mixture of HTML, JavaScript, PHP (or different backend technology such as JSP or .NET), AJAX etc.

Traditional data flow testing when applied to the web applications is considered as object based data flow testing [4]. Objects are the data flow information of the application, and each object contains certain attributes. This is coupled with structural model and the control flow graph. The objects are monitored and tests are defined based on creation (definition) of those objects and their later usage. Again, there are some restrictions, for example not all requests and responses, or navigation between the pages can be represented in this model.

Black box testing

As in traditional testing, black box testing of the web application is completely based on the requirement specification document, and actual access to the code is not assumed. In fact, it is not required to check either code, structure, or any other implementation details of software under test at all.

The most suitable black box method for web applications is the Finite State Machine (FSM model). Naturally, this technique is suitable just for software which behavior can be represented with the finite state, which is usually the case of the web applications. Model is defined with states, transitions between states, and events and actions that trigger those transitions. Events are always triggered with some user input, and actions are responsible for producing some output. This model is usually shown with state transition diagram [5]. For example, state transition diagram for application which is performing online shopping, from the client side, would look like it is shown in Fig. 5 (traditional example in software testing literature).

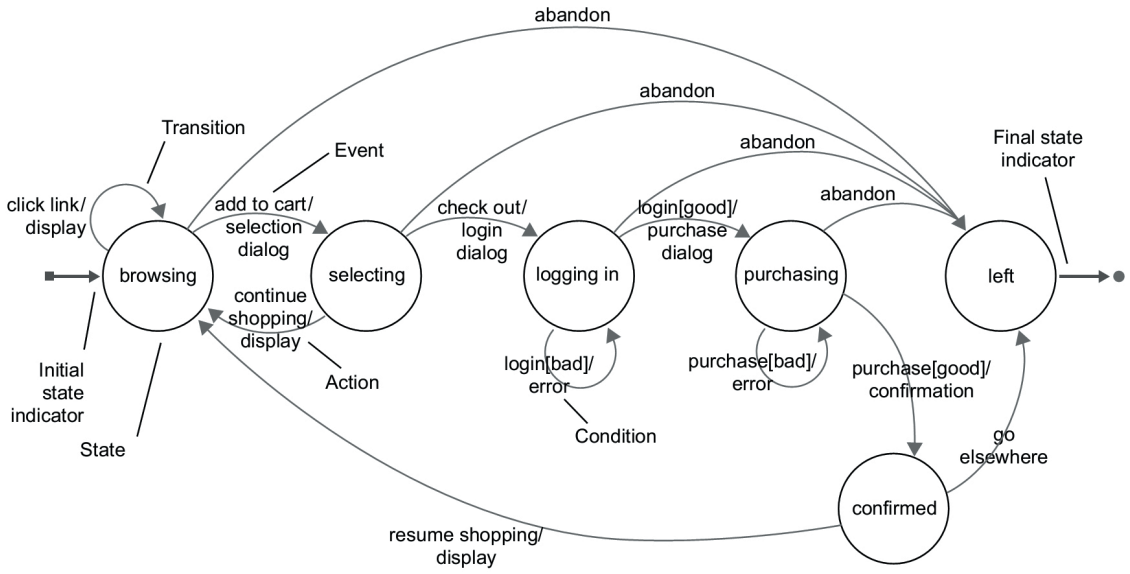

Fig. 5. FSM state transition diagram for online shopping

There are several ways of generating test cases based on the state transition diagram. Criteria for generating test cases from the state model can be one of the following:

- State cover – test set must visit every state in diagram at least once during execution
- Transition coverage (0-switch cover) – test set must exercise every transition in the diagram at least once
- Switch coverage (1-switch cover) – test set must exercise every possible pair of transitions (sequences of two consecutive transitions) in diagram at least once)
- N-switch cover – test set must exercise every possible sequence of N + 1 transitions in diagram at least once

Although every web application can be modeled with FSM, sometimes it can lead to state space explosion problem. Several methods were proposed to fix this issue, such as using dependency analysis in model checking control flow graph [6], or exploiting decision table as a combinatorial model [7].

Other traditional black box techniques commonly used in testing web applications are equivalence class partitioning [8] and boundary values analysis [9]. These two techniques can be applied when testing forms which accept user input. For example, let us observe input field where user should enter his or hers age, and let us assume that, as defined in the specification, user must have between 18 and 99 years, as shown in Fig. 6. We can identify three equivalence classes, one legal (age between 18 and 99), and two illegal, (age < 18 and age > 99). Equivalence classes partitioning is usually coupled with boundary values analysis, as the most probable place where developer could make an error is the processing of the actual border between two equivalence classes. In this case, for example, tester would need to verify the behavior of implemented input validation for entered age 17, 18, 19, 98, 99 and 100, if the stronger border check is applied.

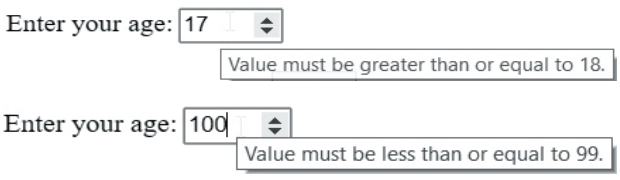

Fig. 6. Border value analysis for the input fields

Additionally, it must be verified whether the validation of the input fields is implemented correctly by checking what happens when unexpected input is entered, such as if random text with characters is entered in the field which should accept only numbers according to the specification. If this validation fails, it is very possible that the error will be generated at the backend, or even propagated to the database where it could trigger database error. This validation could be partly done

with the HTML 5 itself, but additional verification and input validation and sanitization should be done from the JavaScript side as well, and possibly on the back-end, before triggering any actual SQL query toward the database. As discussed earlier, any available input field should be tested in order to verify that it is not possible to enter some malicious input such as SQL injection, which could lead to serious issues with database. This is especially necessary for any application operating with sensitive data, which could lead to any privacy data leak, or financial loss, or in the worst case scenario, even life loss.

4. CONCLUSION

With the raising popularity of web applications, and taking into the consideration complexity and number of different technologies needed to build a web application, web application testing plays a key role. In this paper, we focused on the challenges mostly on the frontend side, which is visible on the client web browser, and easily accessed by the tester. Even this part of the web application brings several problems which must be addressed. For example, FSM is commonly used for black box testing of web applications, but state explosion must be avoided. This can be done either by employing additional methods on the testing side, or by keeping clean and simple design from the development side, which is not always possible.

In this paper, we tried to cover similarities and differences between traditional approach for testing desktop applications, and the needs for testing web applications. Main conclusion can be summed in the fact that traditional black box techniques can be used with small adaptations, but that white box techniques are difficult to be applied to the web application as a whole. White box testing is fully dependent on the actual implementation technologies, and any future technique developed specially for web applications must adapt to the dynamic nature of web application and to the fact that web application is usually built with several different technologies.

REFERENCES

[1] https://www.istqb.org/references/istqb-web-mobile-apps.html

[2] https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[3] Marchetto, A., Ricca, F., and Tonella, P. (2008a). A case study-based compari-son of web testing techniques applied to ajax web applications. Int. Journal on Software Tools for Technology Transfer, 10(6):477-492.

[4] C. Liu, D.C. Kung, P. Hsia, C. Hsu, Object-based data flow testing of Web applications, in: Proceedinds of the First Asia-Pacific Conference on Quality Software, IEEE Computer Society Press, Los Alamitos (CA), 2000, pp. 7–16.

[5] http://istqbexamcertification.com/what-is-state-transition-testing-in-software-testing/

[6] Park,S., Kwon,G.: Avoidance of State Explosion Using Dependency Analysis in Model Checking Control Flow Model. ICCSA (5) 2006: 905-911

[7] G. A. D. Lucca and A. R. Fasolino, --Testing Web-based Applications: The State of the Art and Future Trends|, Information and Software Technology, vol. 48, 2006, pp. 1172-1186.

[8] http://istqbexamcertification.com/what-is-equivalence-partitioning-in-software-testing/

[9] http://istqbexamcertification.com/what-is-boundary-value-analysis-in-software-testing/