Generating Test Cases With High Branch Coverage for Web Applications

Andrey Zakonov and Anatoly Shalyto
National Research University of Information Technologies,
Mechanics and Optics, Saint-Petersburg, Russia
Email: andrew.zakonov@gmail.com, shalyto@mail.ifmo.ru

***Abstract*—Web applications have become significantly more complex and have begun to be used in wide variety of areas including social networks, shopping, online banking and other safety critical systems. We present an approach for automated white-box test generation for web applications. Our approach is to convert problem of high branch coverage test suite generation into a reachability problem and to utilize existing software verification techniques to generate test data for each execution branch. Set of Abstract Syntax Trees (ASTs) is built that describes web application as a whole, both client- and server-side code, by analyzing JavaScript code, its AJAX server calls and callbacks. Constructed AST is converted into an C# function with similar behavior and a set of arguments that represent user inputs. Existing test automation tools are used to discover test data that covers all the possible execution branches in a C# function. Web application test cases are generated with the discovered values, Selenium toolkit is used to emulate user actions and to automatically run the program under test against test cases.**

I. Introduction

Over the recent years web applications have begun to be used in wide variety of areas and have become safety critical, as often contain personal or financial information. Nevertheless relatively little tool support is available to assist testing of web applications. Web applications are usually tested by manually constructed test cases using unit testing tools with capture-replay facilities such as Selenium [1] and Sahi [2]. These tools provide functionality for recording the GUI actions performed by a user in test scripts, for running a suite of tests, and for visualizing test results. However, even with such tool support, testing remains a challenging and time-consuming activity because each test case has to be constructed manually.

As a web application we consider a set of pages connected by hyperlinks, ranging from a set of html pages displaying static content to a complex single page applications, which Document Object Model (DOM) could be dynamically modified by the application logic implemented as JavaScript handlers for user interactions and AJAX callbacks. Originally designed for simple scripting, modern JavaScript programs are complex pieces of software that interact with users and servers and play central role in a web application.

Because of web applications' client-server model, asynchronous server communication and event-driven nature tools and approaches developed for structured and object oriented programs can not be applied out-of-the box for web applications quality assurance. In [4] we addressed problem of creating models of web applications and proposed to use random-driven approach to cover different web application states. In [3] a method to apply Genetic Algorithms to test generation problem for a given EFSM was proposed. In the current research we develop an approach that automates process of test cases generation for a given web application state and generates a test suite with high branch coverage.

There are approaches that generate test cases for structured programs with high branch coverage [5], [6]. For web applications task becomes way more complicated and existing approaches could not be applied as is. Business logic is divided into client-side and server-side code. These two pieces of code are completely different, are implemented using different programming languages, frameworks etc. Communication is done using only HTTP requests. Current research aims to overcome this issues and to propose a method that would be able to adopt existing techniques to achieve high branch coverage for web applications. We analyze application JavaScript code to discover set of possible user actions as well as user inputs these actions depend on. JavaScript action handlers are analyzed and all dependencies of webpage elements and user inputs are extracted from source code and treated as function arguments. Then each possible action could be treated as a JavaScript function call with number of arguments.

If a JavaScript function makes AJAX calls to server-side code, then client-side and server-side code are treated as a combined function. Abstract syntax trees are constructed both for the client-side functions and for the server-side function. While it is not possible to automatically convert code from one language to another, it is possible to construct ASTs for JavaScript handler and callback, as well as for the server-side code, and to replace AJAX request node with a subtree that describes the server-side source code. Resulting AST would describe client-side code together with server-side code and could be represented as one function with a set of arguments. Given a function that emulates behavior of the web application problem of the test generation could be treated as a reachability problem and exiting tools to discover test data could be applied.

The rest of this paper is organized as follows. Section II contains a brief overview of the existing approaches and tools for web applications testing. In Section III the proposed approach is presented. Detailed AST generation and post-processing is given in Section IV. Section V tells about

developed and utilized tools that automate described approach. An example of the code where presented approach could be useful is presented in Section VI. Section VII describes current limitations of the approach and gives an overview of the future research plan.

II. Related Work

There are number of approaches and tools aimed at testing web applications, but none of them is able to automatically generate tests cases that would test the web application as a whole. Existing tools are designed to test some part of the application but not the web application as a whole, which consists of an GUI, client-side JavaScript, server-side code and some tricky interaction between these parts. In the proposed approach we make an attempt to combine different existing tools functionality and to create a framework that would be able to test web application as a whole and to generate test cases automatically. Current section describes existing tools, state their pros and cons and briefly explain how some of these tools would be utilized in the proposed approach.

Web application is usually an event driven system where user generates events. Selenium[1] and Sahi[2] tools could be used as a driver that provides facilities to emulate user interaction with a web page, that include filling out forms with user input and mouse clicks. These tools can be used to play given sequence of actions, but not to generate these sequences. Manual generation of these sequences could be useful for regression testing but it is impossible to achieve high code coverage using them manually. In proposed approach we generated these sequences automatically and utilize Selenium tool to execute generated test cases.

In [8] a framework for automated testing of JavaScript web applications is proposed. The goal of that research is to develop scalable and effective algorithms for automated testing of JavaScript applications to discover common programming errors of various kinds, including uncaught exceptions, improper type coercions or misuses of the browsers built-in libraries, invalid HTML, AJAX HTTP errors, and violation of assertions. While described tool seems to be useful for analyzing source code of the client-side, which is implemented in JavaScript, it does not take into consideration server-side code of the web applications as we try to do in our approach. That would be a serious drawback for complex applications, as it is common to implement most of business logic of the application on the server side. Moreover behavior of client-side code can not be analyzed correctly as it depends on AJAX callbacks which result could be estimated only if server-side code is analyzed.

Other existing tools for testing JavaScript of the web application are mentioned in [8]. The Crawljax/Atusa [9] project uses dynamic analysis to construct a model of an applications state space, from which a set of test cases is generated. The Kudzu project [10] combines the use of random test generation to explore an application's event space with the use of symbolic execution for systematically exploring an applications value space (i.e., how the execution of control flow paths depends on input values). Kudzu relies on an elaborate model for reasoning about string values and string operations, which are frequently manipulated by JavaScript applications.

There are number of researches [5], [11], [3] that address problem of generating tests with high branch coverage for common applications code that does not have specifics like client-server interaction. In proposed approach we treat server side-code as a common application with a specified entry point and set of arguments. Existing tools and frameworks are utilized to generate set of input values to cover all possible code branches in web application source code.

III. Proposed Approach for Generation of Test Suites with High Branch Coverage

Our approach is to convert the test case generation problem into a reachability problem. We observe that a web application can be described by its DOM state and set of possible user actions and corresponding JavaScript callbacks. Therefore problem of testing web application as a whole is separated into two tasks:

1) Discover all possible states of the web application.
2) For each discovered state cover each possible action and callback with a test case.

In [4] we addressed problem of creating models of web applications and proposed an approach to discover all possible different web application states. In current research we address the second task. For a given DOM state of the web application set of tests need to be generated that would cover all feasible execution paths. To achieve this goal we require a model of the system that describes all possible execution paths considering both client- and server-side code. In the developed approach we build an AST that describes control and data flow of the web application as a whole.

Proposed approach consists of the following steps:

1) Analyze client-side source code of the web application and build an AST that describes all possible JavaScript code branches.
2) For each graph node that contains an AJAX call to the server find corresponding server-side function. Source code of the server-side function is analyzed and also an AST is built.
3) Each node with AJAX call is replaced with a corresponding constructed server-side AST.
4) Final AST that describes the system in whole is post-processed to discover set of user inputs that conditional branches depend on.
5) AST is then represented as a C# or Java function that has user inputs as arguments and all condition branches are presented in its body. From this point of view problem of web application testing becomes problem of testing a common function with a set of arguments. Existing technologies and tools are utilized to generate test data that provide high branch coverage for the given function.
6) For each feasible execution path a Selenium [1] test case is generated where discovered test data is used as a user

input that would make the web application cover the desired branches.

7) Application specific business logic test oracles could be optionally added to the generated test cases.

Developed approach makes it possible to automate generation of a test suite with high branch coverage for a given web application state. Combined with earlier proposed state discovery algorithm [4] it is possible to generate test cases that cover web application source code in whole. Generated test suite would be able to discover common programming errors, including unhandled exceptions, runtime errors, undefined variable dereferences, invalid function calls, violation of assertions, invalid DOM operations and etc. Application specific business logic test oracles is impossible to generate automatically and need to be added by developers in the form of JavaScript or server-side assertions that would be also checked automatically during test execution.

Detailed description of the approach steps, problems that were solved and developed tools are presented in the further sections.

IV. Constructing a Simplified AST that Describes a Web Application as a whole

To cover web application's source code with tests a representation is required that describes both client-side code and server-side code in a unified form. In general, there is no existing technique to convert source code from one programming language into another. Moreover, while client-side business logic is always implemented in JavaScript, there are wide variety of programming languages and frameworks that could be used for server-side implementation, which makes task of converting all the source code to one language even more infeasible. But for the purpose of testing higher level of abstraction could be enough, so both server and client-side source code could be converted to some common representation that would be enough for the given task. We propose to use an Abstract Syntax Tree representation for this task.

Basically to cover all possible code branches with test cases a Control Flow Graph of the application should be analyzed. In a control flow graph each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block [13]. But in practice such CFG representation would not be enough to generate test data automatically because it lacks data flow information. Condition in the flow graph nodes depend both on global and local variables. Such variables could be arguments to the function (like user inputs that are not modified during the execution), but also variables could be introduced locally that stand for loop counters, post-processed argument values and etc. Basic blocks of CFG could contain statements that modify such variables and excluding such information from the model would make analysis impossible.

Another way to describe source code using higher level of abstraction is Abstract Syntax Tree. AST is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is 'abstract' in the sense that it does not represent every detail that appears in the real syntax [17]. While CFG does not contain all the required information for the analysis, AST often contains superfluous information. Code statements that do not affect data flow of control flow of the program should be left out of the scope of the analysis.

In our approach we build a simplified AST that describes the application behavior. An algorithm is developed to construct an AST that describes web application page in whole:

1) Gather all the JavaScript source code in one place. Normally a web page's JavaScript source code could be divided into different .js files, that are included in the header block, as well as located in different parts of the HTML file. All occurrences of JavaScript code are discovered by searching for the pattern: `<script src=""></script>`
2) Build AST for the given JavaScript code. In practice it would be set of ASTs, as some parts of code will not be connected to other. For each separate piece of code an AST is built.
3) For each AST find an event that triggers corresponding code. JavaScript code is mostly event driven. Part of the code is executed on page load, but, usually, most of the code is located in action handlers and in callbacks for server calls. For each root node in AST a trigger is defined: page event, like timer or on load, user action or a server callback.
4) Locate all nodes that represent AJAX server calls. Server url information can be extracted from this node, as well as callback function. While in some cases server side function could be determined automatically, there are cases when automatic detection by given url is impossible (complex nginx configs, url mapping, redirects and etc.) and developer should provide such mapping manually to assist test data generation. An AST is built for server-side code and corresponding node in JavaScript AST is replaced by the constructed AST.
5) While arguments list passed from JavaScript directly maps to server function arguments, server-side arguments' names may differ from client-side code names. To preserve correct data flow server-side ASTs is post-processed and all names are replaced with its corresponding JavaScript names.
6) Each return node in the server-side AST is replaced with the corresponding callback AST. If the server-side call returns a value then function's result is introduced as a local variable and a corresponding assignment operator is added.
7) DOM access operators that used to fetch user input are replaced with function arguments. `"replacedN"` is introduced as a function argument and DOM access is replaced with variable access. For further generation of Selenium test sequences an object is created which

maps introduced variables to the actual objects in the following form:

```
{"replaced1":"document.getElementById(
'myTextInput').value()",
...,
"replacedN":"$('.ageInput').val()"}
```

When generating Selenium test cases arguments to the examined function are passed by filling corresponding form elements with discovered by the constraint solver values.

8) All superfluous nodes are removed. If a node does not affect data flow or control flow of the application then it is unimportant for the test data generation process and could be removed. JavaScript code normally contain a lot of DOM manipulation operators that are used to update page correspondingly to the new state. All these operators are useless on the desired level of abstraction and therefore are removed.

A set of ASTs is built after the completion of the listed steps. Constructed set describes web application page in whole, its data and control flow both for client-side and server-side functions. Having such description of the web page problem of test data generation can be converted into reachability problem. Each path in the AST is described as a list of constrains (conditions from the conditional branches) and a set of arguments that have to satisfy these constrains for the execution to follow the selected path.

List of constraints and arguments need to be analyzed to discover unsupported items. Server-side code implements business logic and often contain external dependencies like:

- database resources;
- file system;
- session variables;
- external web services calls.

For paths in the AST that contain any of the listed dependencies automatic test cases could not be generated, as tests would not be able to emulate state of the external objects. Only paths that do not have such dependencies would be a valid input for the constraint solver. Listed external dependencies are a strong limitation for any approach that attempts to automate testing, and ours is not an exception.

We propose to solve this problem by manually introducing mock objects that would emulate database, file system and other dependencies. Our tool could assist in creation of such objects by automatically creating list of required APIs for these mock objects. Developer would be provided with the generated class interfaces he would implement manually.

V. Tools to implement proposed approach

Goal of the proposed approach is to automate testing of web applications. Developed approach consists of number of different steps and for each step a special tool is required to automate the process. While for some steps existing tools could be utilized for other steps proof-of-concept tools were developed. A proof-of-concept framework that implements the proposed approach is being developed using Python 2.7 and JavaScript programming languages.

Static analysis of the web page is performed using a developed tool, the utilizes Selenium framework functions [1] and LXML python library [19] to parse web page and to build a DOM tree. Earlier developed tools [4] are utilized to discover and to compare web application states. Client-side JavaScript code is analyzed using Treehugger library [12], which is able to build an AST describing source code. Treehugger also supports usage of jQuery framework in JavaScript code that is often used to manipulate web page's DOM.

Proposed approach would suite for any language used for server-side development, the only requirement is that an AST with the correct syntax could be constructed for this language. Currently PHP and Python languages support were added during the research. ASTs for the PHP code are built using PHP-Parser [14]. Python source code AST representation could be retrieved using a built-it module [15].

Once all ASTs are brought to the common format merging and post-processing is performed by a developed python tool: superfluous nodes are removed, DOM inputs access are replaced by function arguments, AJAX calls and callbacks are replaced by the corresponding ASTs. All ASTs are stored in text files in the form of string representation.

In order to generate test data that would cover all execution branches we utilize concolic testing tools. Concolic testing is a hybrid software verification technique that interleaves concrete execution (testing on particular inputs) with symbolic execution, a classical technique that treats program variables as symbolic variables. Symbolic execution is used in conjunction with an automated theorem prover or constraint solver based on constraint logic programming to generate new concrete inputs (test cases) with the aim of maximizing code coverage [18]. There are list of tools that implement described technique for different programming languages: PathCrawler, PEX, DART, CUTE. We propose to use Microsoft Pex that is publicly available as a Microsoft Visual Studio 2010 Power Tool for the NET Framework [5]. PEX tool supports C# source code therefore final AST set need to be converted to C# source code. A converter is being developed during the research to perform this conversion using a straightforward algorithm.

Once test data is generated test case for each data set are generated. Test cases are implemented in python language and utilize Selenium WebDriver, a collection of language specific bindings, to emulate user actions and inputs in the given browser. PyUnit tool can run the generated test suite automatically.

VI. Illustration of the proposed approach

Proposed approach could be illustrated with an analysis of a part of the website registration form. User inputs are often checked at server-side code for different business logic constrains. For example a valid value for height of a new user has to be more than 50cm and less than 250cm. Client-side code contains an `input` form element and for its `change` event a jQuery handler is introduced. Inside handler an AJAX

call to server is made and entered value is validated with the following PHP code:

```
if (v > = 50 and v <=250) {
return "true";
} else {
return "false";
}
```

JavaScript callback parses returned value and displays an error message if `"false"` was returned. For the given example a combined AST is generated:

```
Function( "handler",[FArg("v")],[ If( Op(
"&&", Op(">=",Var("v"),
Num("50")),Op("<=",Var("v"),Num("250")
)),Block(...),Block(...))])
```

PEX tool is used to find input values for two possible branches with the following constrains:

1) `v >= 50 && v <= 250`
2) `!(v >= 50) || !(v <= 250)`

Test data generator easily finds values that suit for both execution paths and knowing that `v` stands for `$("#height-input").val()` page element two Selenium tests that would trigger two possible execution paths are generated.

VII. Conclusion

Web applications are built using client-server model and operate with callbacks rather than then sequential method calls. For these reasons tools and approaches developed for structured and object oriented programs can not be applied out-of-the box for web applications quality assurance. In this research we made an attempt to adapt existing test techniques to support web applications specifics. For each possible user action combined syntax tree is constructed to describe client-side and server-side source code together. Existing concolic testing tools are applied to the generated source code that behaves similar to web application. Discovered test data and Selenium WebDriver are used to generate a set of test cases with high branch coverage for the given application.

In the further research it is planned to develop a strategy that would assist developers in creating mock objects to make it possible to test functionality that depend on external objects like database and file system. Asynchronous nature of web pages interaction with server-side code also needs further investigation. Nevertheless proposed testing technique could be used to automate testing of the web applications and significantly improve software quality and defect detection rate.

References

[1] Antawan Holmes , Marc Kellogg, Automating Functional Tests Using Selenium, AGILE 2006: .270–275

[2] Web test automation tool. http://sahi.co.in/w/sahi

[3] Zakonov A., Stepanov O., Shalyto A.: GA-Based and Design by Contract Approach to Test Generation for EFSMs. IEEE EWDTS 2010: 152–155.

[4] Zakonov A., Shalyto A.: Automatic Extraction and Verification of State-Models for Web Applications. Lecture Notes in Electrical Engineering, 2012, Volume 133, Part 1, 157-160

[5] Tillmann N., Halleux J.: Pex – White Box Test Generation for .NET. Lecture Notes in Computer Science, 2008, Volume 4966/2008, 134-153,

[6] Pandita R., Xie T., Tillmann N., Halleux J.: Guided test generation for coverage criteria. ICSMIEEE Computer Society (2010) , p. 1-10.

[7] Bartak R., Salido M., Rossi F.: New Trends in Constraint Satisfaction, Planning, and Scheduling: A Survey. The Knowledge Engineering Review, 2004, Cambridge University Press

[8] Artzi, S., Dolby, J., Jensen, S.H., Moller, A., Tip, F.: A framework for automated testing of javascript web applications.In ICSE(2011)571-580

[9] A. Mesbah and A. van Deursen. Invariant-based automatic testing of AJAX user interfaces. In Proc. 31st Int. Conf. on Software Engineering, ICSE 09, May 2009.

[10] P. Saxena, D. Akhawe, S. Hanna, S. McCamant, D. Song, and F. Mao. A symbolic execution framework for JavaScript. In Proc. 31st IEEE Symp. on Security and Privacy, S&P 10, May 2010.

[11] Arcaini, P.; Gargantini, A.; Riccobene, E.: Optimizing the automatic test generation by SAT and SMT solving for Boolean expressions. ASE, 2011 26th IEEE/ACM.

[12] JavaScript AST library. https://github.com/ajaxorg/treehugger

[13] Control flow graph. Wikipedia. http://en.wikipedia.org/wiki/Control_flow_graph

[14] PHP-parser and AST builder. https://github.com/nikic/PHP-Parser

[15] Built-in module for AST in Python. http://docs.python.org/library/ast.html

[16] Python module offering solvers for Constraint Solving Problems. http://labix.org/python-constraint

[17] Abstract syntax tree. Wikipedia. http://en.wikipedia.org/wiki/Abstract_syntax_tree

[18] Concolic testing. Wikipedia. http://en.wikipedia.org/wiki/Concolic_testing

[19] Feature-rich and easy-to-use library for processing XML and HTML in the Python language. http://lxml.de/