# Test Cases Generation Tool for JavaScript Based on Statement Coverage Criteria

Witthaya Luanghirun, Taratip Suwannasart

***Abstract*—In modern web application development, JavaScript is the most popular programming language for implementation and test framework automation is usually applied in unit testing. However, developers spend a lot of time to create test script manually. Thus, creating automated test script tool can support them efficiently. Nonetheless, a tool for generating test script by randomly creating test input cannot guarantee that all paths of the code is executed and it takes significant of time on testing to reach a high code coverage. This paper proposes a tool for generating test cases from JavaScript function and executing test cases to cover all statements coverage criteria. The tool can analyze and instrument JavaScript code to generate a control flow graph and test cases by selecting data based on test paths and input vector to drive the paths, evaluate coverage, execute test cases, as well as display a test report.**

***Keywords*— Software Testing, Automatic Testing, Input Vector, Path Predicate Expression, JavaScript**

## I. Introduction

JavaScript is the most popular programming language used in web application development because developers usually use JavaScript to improve web browser better performance. Building an enterprise JavaScript web application generally uses an automated test framework. However, developers spend significant of time to generate test script manually before running an automated test suite.

In unit testing, JavaScript code is analyzed and assigned test input by developers before the executing test cases. These activities require a lot of time to operate. Furthermore, if there are many JavaScript functions to be tested, developers have to spend a lot of time to review and create test scripts.

In previous work [1], the research proposed a tool that can automatically generate test module with randomly creating parameters for JavaScript. The tool can reduce time consuming during test script generation. However, generated test script with randomly creating parameters cannot execute in some test paths in a JavaScript Function that causes low test coverage.

Manuscript received November 27, 2015; revised January 9, 2016.

Witthaya Luanghirun is with the Department of Computer Engineering Faculty of Engineering, Chulalongkorn University, Bangkok, 10330 Thailand (e-mail: Witthaya.L@student.chula.ac.th).

Taratip Suwannasart is an Associate Professor at Department of Computer Engineering Faculty of Engineering, Chulalongkorn University, Bangkok, 10330, Thailand (e-mail: Taratip.S@chula.ac.th).

In this paper, we propose a tool to generate test cases by analyzing path predicate expression [2] for automatic JavaScript unit test. The tool can analyze an input JavaScript file, instrument code, create control flow graph [3], and automatically generate test cases by analyzing paths predicate expression that cover all statements on each test path. A value of parameters consisting string, number, and boolean, will be created from conditional statements. Then, a test script in D.O.H. format is generated and test cases are executed. Finally, it will create a test report of test cases which provides information about test function, test parameters value, and test paths.

We have organized the rest of this paper as follows. Section 2 describes related work. Section 3 presents development of our proposed tool. Finally, conclusion is represented in Section 4.

## II. Related work

### A. *Tool for Generating Test Module for JavaScript Based on Statement Coverage Criteria [1]*

This research proposed a tool that automatically generates test module with randomly creating parameters for JavaScript which presents the concept of parameter type determination on JavaScript function and the operation of open-source frameworks in JavaScript unit testing.

Parameter Type Determinator is a module of the tool that provides functions to determine types of parameters. They presented an approach to determine parameter types by inspecting parameter usages in function.

D.O.H. [4] is a component of open-source frameworks, Dojo Toolkit [5], which provide test framework for JavaScript. This tool automatically creates test scripts in D.O.H. format and executes test scripts through the D.O.H. runner. Moreover, D.O.H. can test not only JavaScript within the framework but also normal JavaScript outside framework.

### B. *Computation of the Minimal set of paths for Observability-based statement coverage [6]*

This paper presented finding a minimal set of execution paths that assured a user-specified level of observability-base coverage for embedded software program. This research proposed an application white-box testing to find the paths by building a Directed Acyclic Graph, and generating an input value from feasibility test path due to observability-based coverage. Moreover, the research represented an input vector generation by deriving and evaluating all the branch conditions on the given paths to refine an input vector for desired outcome.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

## III. Implementation of the Proposed Tool

In this paper, we propose a tool to generate test cases by analyzing path predicate expression for automatically unit testing JavaScript. The structure of the proposed tool is presented in Fig. 1.

The tool is divided into four main parts which consist of 1) analysis and instrumentation of the JavaScript code, 2) control flow graph generation, 3) test case and coverage data generation and 4) test cases execution. There are also 4 minor modules in test case and coverage data generation part which consists of path selection, path predicate expression, input vector value generation, and test case and coverage data generation. Each part of the tool is described in steps as shown in Fig. 1.

### A. The JavaScript code Analysis and Instrumentation

Since the tool receives a JavaScript source code from a user. An example code is shown in Fig. 2. The tool analyzes parameters a function as input vectors [2] which are inputs of the function that determine the decision of predicates within the function. Input vectors are determined the data type by inspecting their usages. This tool supports data type of input vector in string, number, and boolean only.

Then, the source code is read and instrumented as shown in Fig. 3. The instrumented code is added string counter statement name "jscounter" for inspecting the path during test execution.

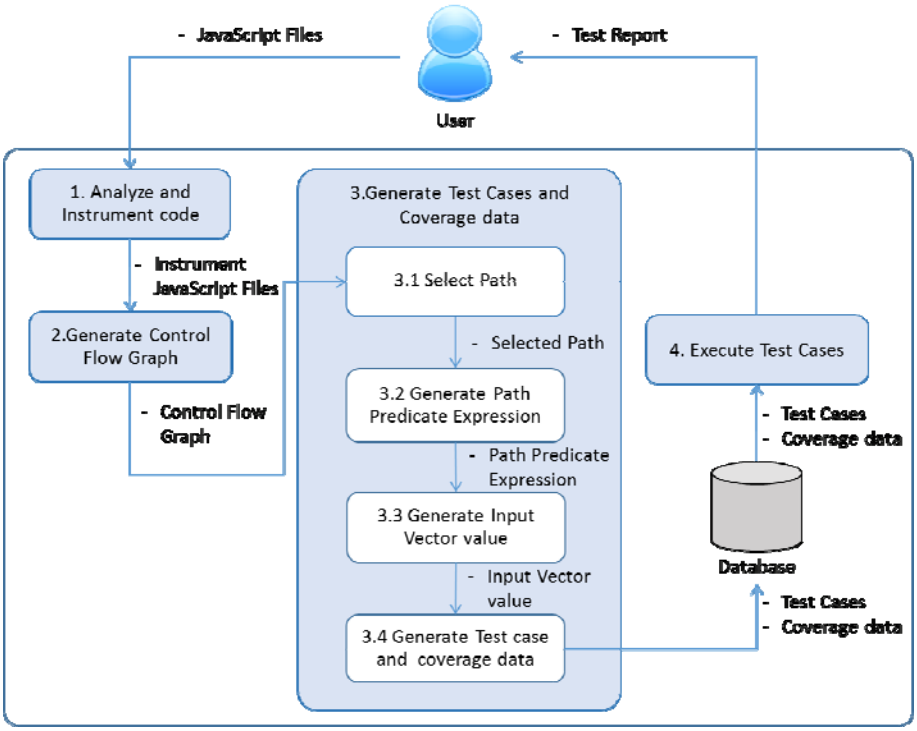


Fig. 1. An Overview of the tool structure

```
1   function main(a,b){
2      var x = a + b;
3      var y = a - b;
4      if (x > y)
5         {z = x + y;}
6      else
7         {z = x - y;}
8      if (z < 0)
9         {z = 0;}
10     return z;
11     }
```

Fig. 2. An example JavaScript code

```
1   function main(a,b){
2      var jscounter ="";
3      jscounter += "1";
4      var x = a + b;
5      jscounter += "-2";
6      var y = a - b;
7      jscounter += "-3";
8      if (x > y)
9         {jscounter += "-4";
10        z = x + y;}
11     else
12        {jscounter += "-5";
13        z = x - y;}
14     jscounter += "-6";
15     if (z < 0)
16        {jscounter += "-7";
17        z = 0;}
18     jscounter += "-8";
19     console.log(jscounter);
20     return z;
21  }
```

Fig. 3. The instrumented code of an example code

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

### *B. Control Flow Graph Generation*

After code instrumentation, this process extracts the code and creates a control flow graph. In order to extract paths for generating input vector, each node in the graph represents a JavaScript statement and each edge in the graph is used to represent a flow of the program.

From the given code in Fig. 2, a control flow graph is created as shown in Fig. 4.

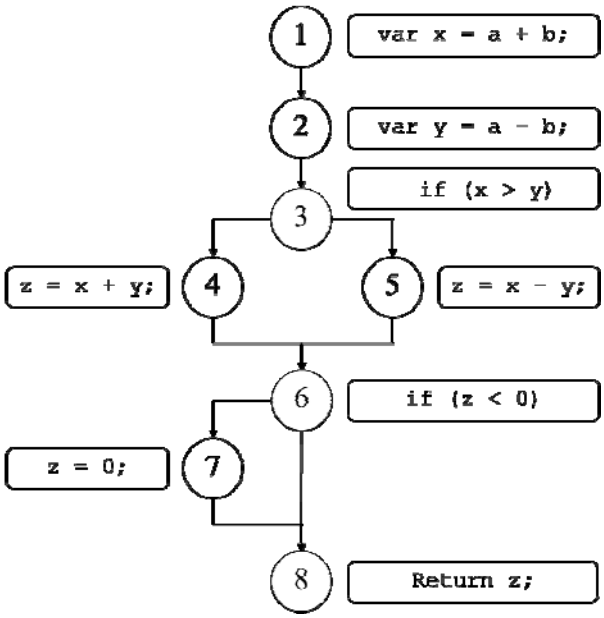


Fig. 4. A control flow graph of an example code

### *C. Test Cases and Coverage data Generation*

This component provides the process to select test paths, creates a path predicate expression for generating test vector values, and creating test cases and calculating coverage data.

#### *1) Paths Selection*

After a control flow graph is generated, the tool will now select paths on a graph by using the depth first search algorithm [7] from the start node through the end node of the graph. For example, in Fig.4 paths through the program which covering each statement are shown in Fig. 5.

1) 1→2→3→4→6→8
2) 1→2→3→5→6→7→8

Fig. 5. Selected paths from Fig. 4

#### *2) Path Predicate Expression Generation*

A Path predicate expression is a path that shows decisions of program statements on each predicate for generating input vectors. In Fig. 4, there are two condition nodes which are node 3 and 6 in the graph. The tool gives a predicate "A" represents the predicate on node 3 which is "x > y" and a predicate "B" represents the predicate on node 6 which is "z < 0". From Fig. 6, a path predicate expression of path no.1 is assigned as the expression "A$\overline{B}$" which the expression "A" represents predicate "x > y" which is TRUE of predicate of "A", and "$\overline{B}$" represents "z ≥ 0" which is FALSE of predicate of "B".

The path predicate expression of path no.1 is A$\overline{B}$
where is A: x > y, $\overline{B}$: z ≥ 0
The path predicate expression of path no.2 is $\overline{A}$B
where is $\overline{A}$: x ≤ y, B: z < 0

Fig. 6. A path predicate expression for selected paths in Fig. 5

After creating the expression, tool interprets the expression to local variable, constant, or input vector which is involved. For example, the expression "A$\overline{B}$" means "A" and "$\overline{B}$" where "A" is "x > y" and $\overline{B}$ is "z ≥ 0". This predicate expression can be interpreted as "(a + b) > (a – b)" and "(a + b) + (a – b) ≥ 0".

#### *3) Input Vector value Generation*

The tool generates input vector values from the given path by consider an input vector type and the path predicate expression. This module analyzes path predicate expression feasibility. The tool will generate input vector if the path predicate expression is feasible. If the given path is not feasible, the tool will return null value.

#### *4) Test Cases and Coverage data Generation*

Since input vectors are assigned with values, the test cases are generated as shown in Table I. A test case consists of attributes as follows:

1. *Test Case ID*: it is assigned after the test case generation.
2. *Test Function*: it represents tested JavaScript function to be tested.
3. *Input Vector*: it represents parameters of a function.
4. *Input Vector value*: it represents values of parameters.
5. *Test Path*: it represents path that will be executed a test from given input vectors.

TABLE I AN EXAMPLE TEST CASE

| Test Case ID | Test Function | Input Vector | Input Vector Value | Test Path |
|---|---|---|---|---|
| TC01 | main | a,b | a : 836<br>b : 37 | 1→2→3→4→6→8 |
| TC02 | main | a,b | a : -568<br>b : -23 | 1→2→3→5→6→7→8 |

The coverage data is presented as percentage of statement coverage of generated test cases which are executed. The formula for calculating percentage of statement coverage is shown as follows:

$$\% \text{ Statement coverage} = \frac{\text{Number of traversed statements at least one time}}{\text{Number of all statements}} \times 100 \quad (1)$$

After test cases and coverage data are generated, they will be stored in a database for using create test scripts and make a test report in the next processes.

### *D. Test Cases Execution*

The tool retrieves the test cases from a database, generates test module and executes test cases in D.O.H.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

framework. An example of test module is shown in Fig. 7.

```
1 define(["doh"],
2 function(doh, test){
3 doh.register("test", [{
4 name: "main_1",
5 setUp: function(){
6   startTest();
7 },runTest: function(t){
8   var res = main(836,37);
9 },tearDown: function(){
10  endTest();
11} },]);});
```

Fig. 7. An example of D.O.H test script

The test script will be executed through the D.O.H. runner and return results. If the results of the test execution match with the test path, it means that the test is passed. Otherwise, the test is fail. Then, the tool provides a test report is shown in Fig. 8.

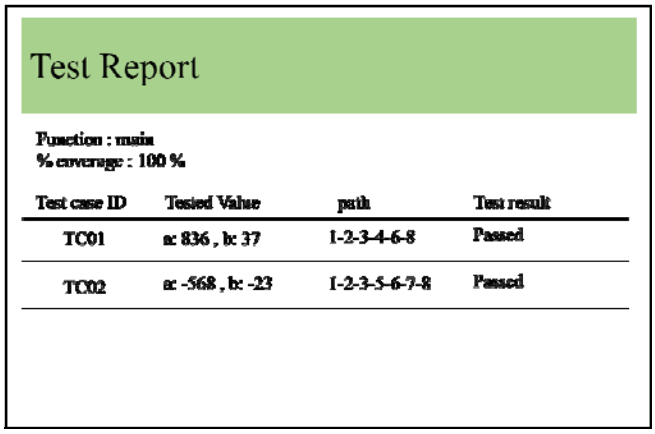


Fig. 8. A Test report

## IV. Conclusion

In this paper, we have presented a tool for automatically generating test cases based on selective parameters for JavaScript function. We present how to generate test data for each test case which support only string, number and, boolean type. The benefit of this tool is the reduction time and effort for creating JavaScript test cases.

## References

[1] P. Janthong and T. Suwannasart, "Tool for generating test module for JavaScript based on statement coverage criteria," in *Computer Science and Software Engineering (JCSSE), 2014 11th International Joint Conference on*, 2014, pp. 331-336.

[2] K. Naik and P. Tripathy, *Software Testing and Quality Assurance Theory and practice*. Hoboken, New Jersey: John Wiley & Sons, 2008.

[3] M. Pezze and M. Young, *Software testing and analysis : process, principles, and techniques*. Hoboken, New Jersey: Wiley, 2008.

[4] The Dojo Foundation. *D.O.H.: Dojo Objective Harness* [Online]. Available: https://dojotoolkit.org/reference-guide/1.9/util/doh.html

[5] The Dojo Foundation. *The Dojo Toolkit* [Online]. Available: http://dojotoolkit.org

[6] J. Costa and J. Monteiro, "Computation of the minimal set of paths for observability-based statement coverage," in *Mixed Design of Integrated Circuits and Systems, 2008. MIXDES 2008. 15th International Conference on*, 2008, pp. 587-592.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, Second ed.: MIT Press and McGraw-Hill, 2001.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)