MULTIMEDIA UNIVERSITY ®

TIS3151 - SOFTWARE RELIABILITY & QA

GROUP ASSIGNMENT

CALCULATOR WEB APPLICATION SYSTEM TEST DOCUMENTATION

Prepared by:

No	Name	ID	E-mail
1	Chai Ying Hua	1141328508	yinghuachai0@gmail.com
2	Shubar, Abduelhakem G Abdusalam	1141327949	hakeem95shubr@gmail.com

Master Test Plan Outline

1. Introduction

1.1 Document Identifier

Document Version	IEEE Std 829-2008 System Test Document Version 1.0 approved
Date of issue	2nd September 2017 (Saturday)
Issue organization	MMU SRQA Assignment team
System	Basic Calculator Application v1.0.0
Authors	Chai Ying Hua, Shubar, Abduelhakem G Abdusalam
Reviewer	Chai Ying Hua, Shubar, Abduelhakem G Abdusalam

Authors

Date:

Reviewer

Date:

1.2 Scope and objectives.

This document describes software reliability methodology, software development models and system testing techniques used in web application development to build reliable software and application in quality practices. The external and internal qualities of calculator web application shall be covered with quality factors aspect to meet software requirements and standards. The project attempts to integrate software development process (SDLC) and testing life cycle (STLC) to ensure every single phases conducted are compliance with implicit and explicit requirements.

Degree of team's acquaintance on web development, software usability and project size are taken into consideration on deciding quality assurance activities. Risk analysis and management are conducted to manage internal risk and external risk with identification of procedures with system requirement specification to avoid and minimize potential impact that affects the development process. Time and system resource constraints are identified and controlled to ensure the project is delivered in specific duration.

The system or components are evaluated to maintain consistency of development activities in each phase and examined system requirements has been fulfilled during the process. Operational aspect such as professional standards and coding procedures are integrated with software development methodology to achieve acceptable quality level before system is delivered.

1.3 Internal References.

1. University of New Orleans. IEEE Documentation Standard for Software and System Test 2008. “Section 8 Master Test Plan”. 2008. 34-41. IEEE Xplore. Retrieved from: https://standards.ieee.org/findstds/standard/829-2008.html

2. University of New Orleans. IEEE Documentation Standard for Software and System Test 2008. “Section 9 Level Test Plan”. 2008. 42-58. IEEE Xplore. Retrieved from: https://standards.ieee.org/findstds/standard/829-2008.html

3. University of Connecticut. IEEE Documentation Standard for Quality Assurance Process 2014. “IEEE Std 730-2014 SQAP Outline Annex B”. 2014. 85. IEEE Xplore. Retrieved from: https://standards.ieee.org/findstds/standard/730-2014.html

1.4 System overview and Key Features.

A calculator is small devices with a keyboard and a visual display that perform arithmetic operations with numbers. The application allow user to perform calculations with basic arithmetic operations.

The key features of basic calculator application are as follow:

- **Addition.** The mathematical operation takes more than one input to perform calculation of total amount of these inputs.

- **Subtraction.** The mathematical operation takes more than one input to perform calculation on finding differences between these inputs.

- **Multiplication.** The mathematical operation takes more than one input to perform calculation on repeated addition under specific rules.

- **Division.** The mathematical operation takes more than one input to perform calculation on splitting into equal amounts of parts or group.

The user should possess basic knowledge of using mouse and keyboard along with text editing conventions. A mini basic calculator web application is developed for conducting various testing activities with plans or script in order to refine software quality. The application is build with HTML, CSS, JavaScript with AngularJS framework.

1.5 Responsibilities and Authorities

The project is conducted in order to satisfy requirement in Software Reliability and Quality Assurance subject's coursework and team members shall not exceed two peoples. All the members possess every roles and responsibilities to complete project due to limitation of human resources. The key authorities and responsibility are tabulated as below:

Self organizing Functional Team	
Requirement Engineer	- Produce details specifications - Identify and obtain requirements
Project Manager	- Conduct project management and risk measurement on requirement specifications. - Identify SDLC, STLC, development methodology and master test plan with teams.
Software Developer	- Design architecture of system. - Identify tools and programming languages for system development. - Creating applications for computerized devices - Set up hardware and software infrastructure for development environment. - Writing program codes based on detail specifications with specific standards. - Conduct automate unit test and integration test.
Software QA Engineer	- Conduct System test and Acceptance Test on Alpha Testing. - Define test benchmarking, prepare test cases, conduct test procedures and document test outcomes into reports. - Bug tracking, reporting and escalation. - Coordinate testing effort between development - Ensure every process and product are doing right

1.6 Test Implementation

1.6.1 Testing tools

1.6.1.1 List of Hardware Resources

a. **64-bit Personal Computer.** This machine is used for development and testing activities for this project. The processor is 8x Intel Core (™) i7-6700HQ CPU @ 2.60hz with GPU NVIDIA GEFORCE GTX960M GDDR4. The ram is 16GB.

1.6.1.2 List of Software Resources.

a. Linux Ubuntu 16.04.3 LTS 64-bit. Operating system for the personal computer.

b. IntelliJ PhpStorm Ultimate version. This IDE is used for development and testing activities for this project. The IDE is enterprise solution which able to integrate with automation testing tools and conduct functional and nonfunctional testing in integrated environment.

c. Jasmine Core 2.4.1. Specific Node.js features for Jasmine which enable Karma Server to host web application and execute test code simultaneously.

d. Karma Server v0.13.22. The tools spawn new web server that execute source codes (HTML, CSS, JavaScript) against test code (Jasmine Unit Test) for each browser connected.

e. Karma Jasmine v0.3.8. Node.js adapter for Jasmine testing framework.

f. Karma-firefox-launcher. The tools launch firefox browser on monitoring web application behaviors.

g. Karma-opera-launcher. The tools launch opera browser on monitoring web application behaviors.

h. Node.js v6.11.3 LTS. Open source JavaScript runtime environment for executing JavaScript code server-side.

1.6.2 Testing techniques and methods.

The software development decision framework used by this project is Agile Software Methodology. The mentioned methodology simplified process decisions around incremental and iterative solution delivery with continuous planning and feedbacks. Agile methodology provide flexibility on product modification and adaptivity based on defects and errors produced in each phase and quickly respond to changes.

Agile software development promote principles for product and testing development under which requirements and solutions evolve through the collaborative effort of self-organizing management in this project instead of following a plan. It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change according to result of testing and quality assurance activities.

The **Agile testing** is used to follow the principles of agile software development with extreme programming method **Test-Driven Development** (TDD). The popularized evolutionary approach is introduced in 2003 which combines testing and development in construction phase in SDLC where we write a test case before writing production code. The behavior and functionality of single unit, module or package has to satisfy test case written with refactoring and defensive programming.

The process allow developer think through requirements and design before write functional codes, ensure every single test case and codes generated are not **redundant.** Instead of writing functional code first and then allow QA engineer to generate test case as an afterthought, the developer write small bit of test case correspond with functional codes to covered every single lines and unit in program. The developer cannot proceed to write new

function and add single line of codes when the test case is failed until the codes are refactor to passed the test case.

1.6.3 Testing metrics

This section describe the metrics used in quality assurance activities to support each level of test plans. Non-functional test (NF) and functional test (F) are conducted in to ensure system functionality meet requirement specifications. The testing methods and plans are tabulated as below:

Test Level	**Test Methods**
Unit Test	- Automation Unit Test (F) - Maintainability Testing (NF)
Integration Test	- Recovery Testing (NF) - Portability Testing (NF)
System Test	- Compliance and Conformance Testing (NF)
Acceptance Test (Alpha Testing)	- Usability Testing (NF) - Documentation Testing (NF)

The details of test level and test methods will be discussed in details in section 4.

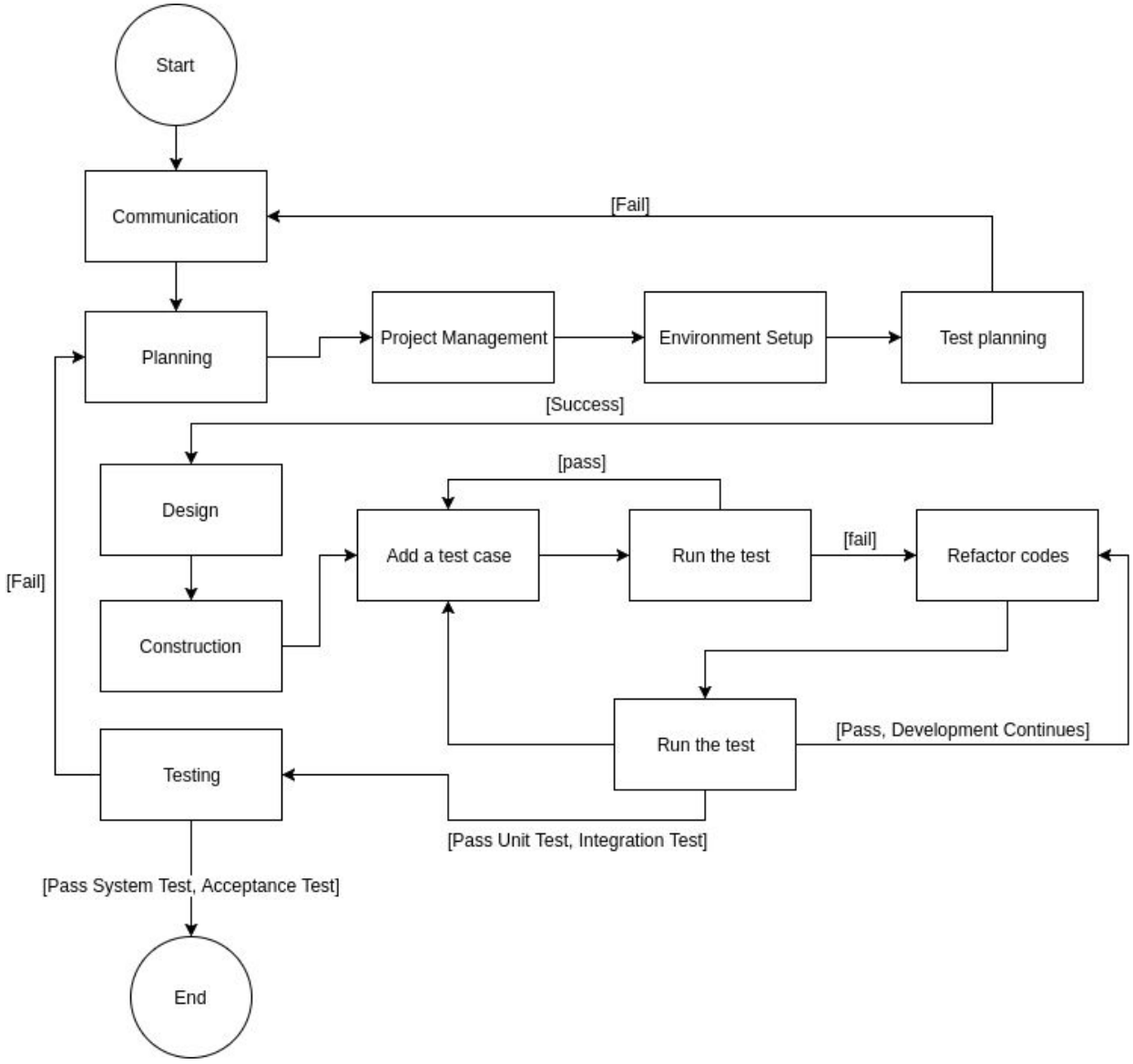
2. Details of Master Plan

2.1 Test management

a. Deliver good detailed project documentation that specifies all functional and nonfunctional requirements of system.
b. Maturity on SDLC, SDTC and agile testing framework selection which concern on time and task pressure in every iterative development process.
c. Skill factors on programming languages, language framework, software testing and development tools, system and hardware infrastructure and standard operation procedure on quality assurance process.
d. Benchmarking and pass/fail criteria for deciding confidence and acceptance level on external and internal qualities of system.

2.2 Testing Designs

The diagram below showss the integration of software testing life cycle, test-driven development with software development life cycle.



2.3 Test tools description.

2.3.1 Karma

Karma is essentially a JavaScript Test Runner, unit-testing web applications which spawns a web server that executes source code against test code for each of the browsers connected. The results of each test against each browser are examined and displayed via the command line or IDE to the developer such that they can see which browsers and test execution is either passed or failed.

There is totally no compiler that can catch syntax error, null object issues, memory distribution, performance compilation and execution in this dynamic language. The karma test runner helps web application developer to be more productive and effective to test project, commonly used by companies such as Google and Youtube.

Karma allow integration of IDE and allow developer to write plugins to extend the testing functionality and enable integration of any framework and library. Moreover, it increase the testability and maintainability which allow developer to debug the web application easily via terminals. The developer is able to inspect call stacks, variable and behaviors of program execution line by line.

2.3.2 Jasmine Behavior-driven Testing tools.

Jasmine is a behavior-driven development framework for testing JavaScript AngularJS framework code. The framework offer readable semantics and syntax which allow automation tester or developer to write test. The figure belows show the test case written with Jasmine.

```
describe('Test Case for Calculator Display', function () {
      it('numberClicked() should display values correctly when user pressed numbers 1-9', function () {

          var $scope = {};
          var controller = $controller('calcCtrl', {$scope: $scope});

          $scope.numberClicked(1);
          $scope.numberClicked(2);

          expect($scope.currentDisplay).toBe("12");
      });

      it('numberClicked() should not display number when user pressed number 0 at first input', function () {

          var $scope = {};
          var controller = $controller('calcCtrl', {$scope: $scope});

          $scope.currentDisplay = "";

          $scope.numberClicked(0);
          expect($scope.currentDisplay).toEqual("0");
      });
  });
```

Listing 1: Automation test case written in Javascript with Jasmine

Jasmine able to conduct integration test with dependency injection with AngularJS, it can easily mocked the dependencies between component to ensure the program is designed and written in defensive manner or information hiding. The testing tools allow to test every units (functions) , components (controller) and relationship between components (package) according to preferences of developer.

2.4 Test tools installation.

A. Install Node.js

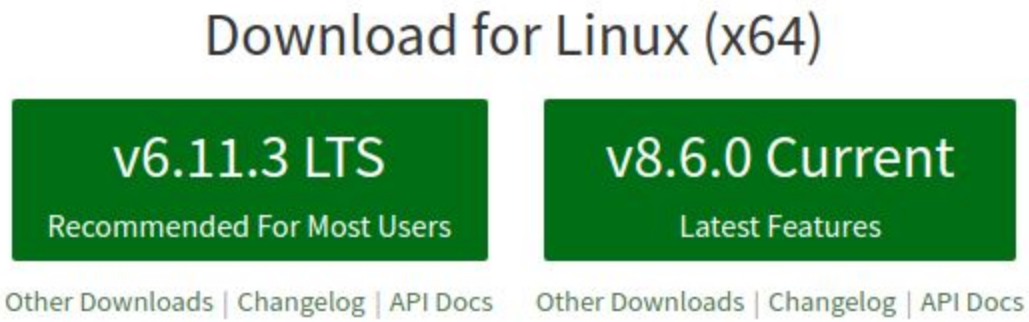


Figure 1: Node.js installation selection

B. Install Karma into PC with terminal

a. Install Karma-jasmine

b. Install Karma-chrome-launcher

c. Install Karma-opera-launcher

d. Install Jasmine-core

```
# Install Karma:
$ npm install karma --save-dev

# Install plugins that your project needs:
$ npm install karma-jasmine karma-chrome-launcher karma-opera-launcher jasmine-core --save-dev
```

Listing 2: Terminal commands for install karma server and their plugins

C. Install Karma plugins for IntelliJ PhpStorm Ultimate

The Karma plugin will be installed in PhpStorm. This can be done from the IDE Settings | Plugins, then clicking Install JetBrains plugin... and searching for Karma. Note that for this plugin to work, the Node.js plugin must be installed.

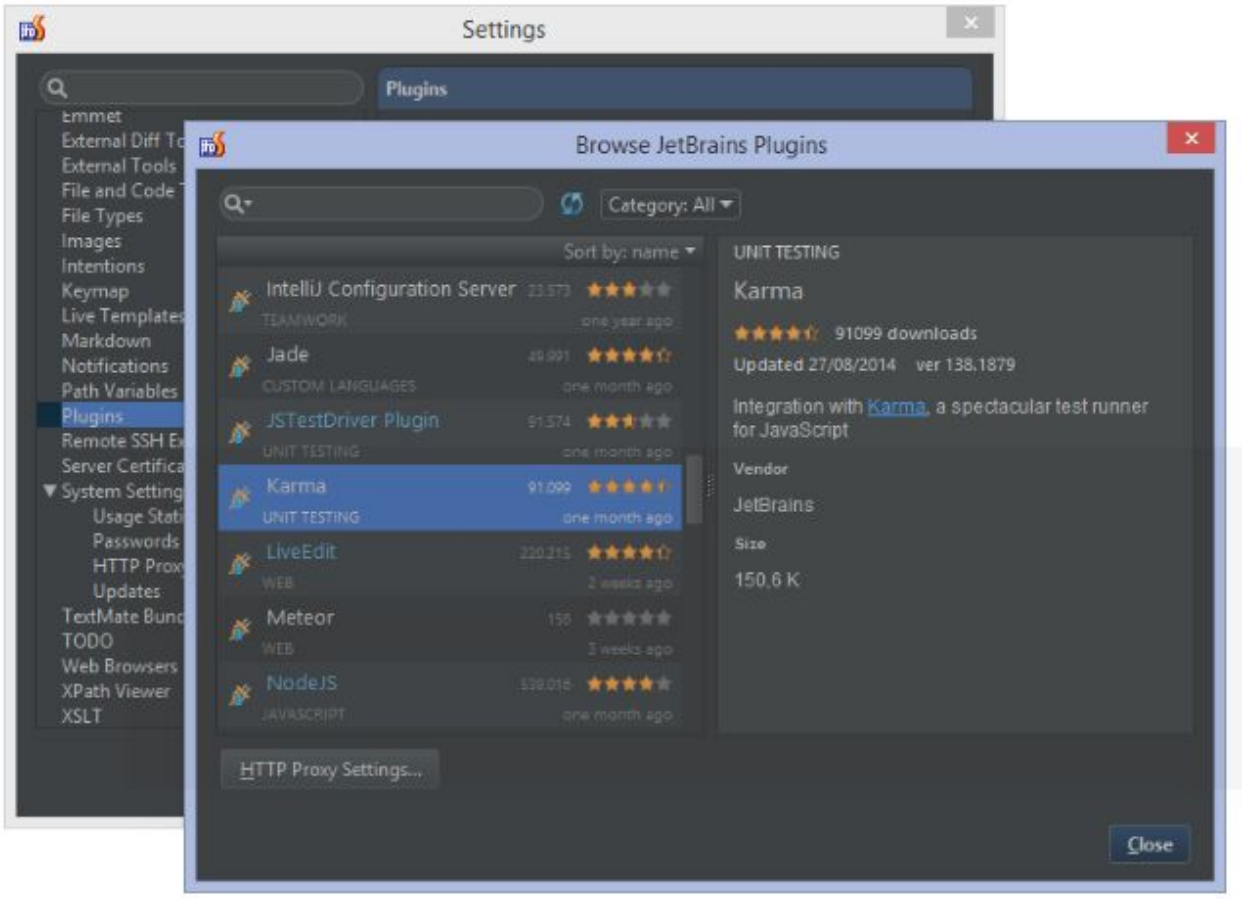


Figure 3: Install Karma plugins into IntelliJ PhpStorm

D. Install Code Coverage in terminal.

```
$ npm install karma karma-coverage --save-dev
```

Listing 3 : install Karma code coverage for testing.

E. Setup Karma configuration with JavaScript. (Karma.conf.js)

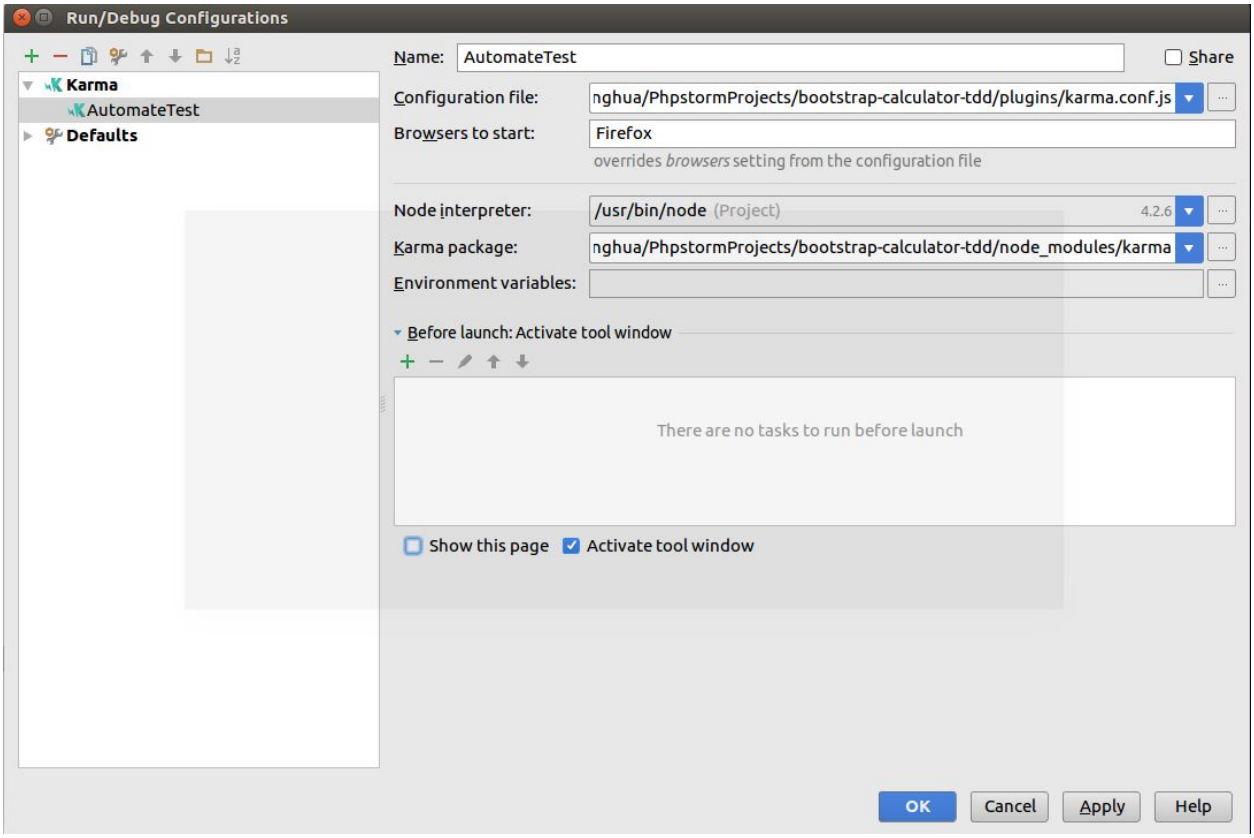
- Include JavaScript path files.
- Include Jasmine frameworks
- Include all the plugins require for testing with Karma and Jasmine.
- Include code coverage into program lines and functions.

```
//jshint strict: false
module.exports = function (config) {
  config.set({

    basePath: '../app',

    files: [
      '../app/angular-sample-app/bower_components/angular/angular.js',
      '../app/angular-sample-app/bower_components/angular-route/angular-route.js',
      '../app/angular-sample-app/bower_components/angular-mocks/angular-mocks.js',
      '../app/calculator-app/calculator.js',
      '../app/calculator-app/calculatorSpec.js'

    ],

    autoWatch: false,

    frameworks: ['jasmine'],

    browsers: ['firefox','opera'],

    plugins: [
      'karma-opera-launcher',
      'karma-chrome-launcher',
      'karma-firefox-launcher',
      'karma-jasmine',
      'karma-junit-reporter',
      'karma-coverage'
    ],

    preprocessors: {
      // source files, that you wanna generate coverage for
      // do not include tests or libraries
      // (these files will be instrumented by Istanbul)
      '**/*.js': ['coverage']
    }

  });
};
```

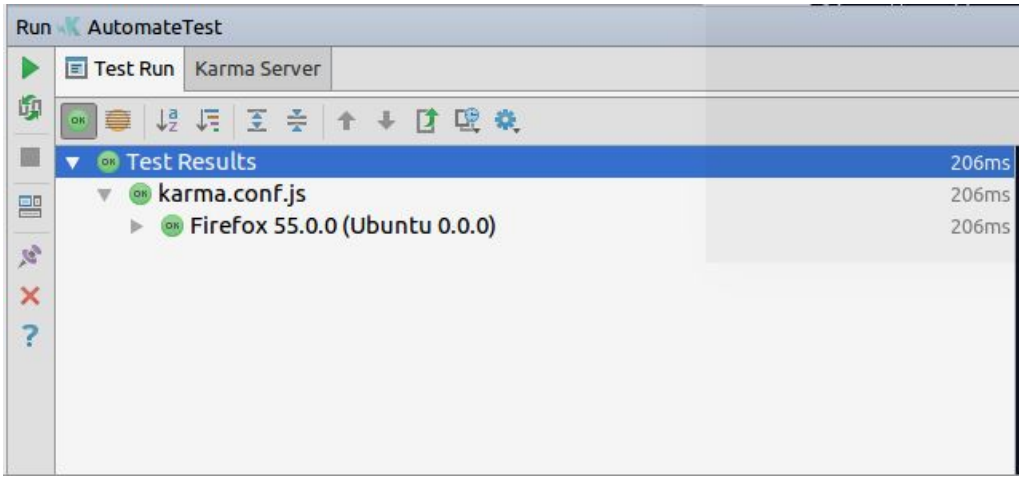
Listing 4: Karma.conf.js sourcecodes

F. Setup configuration on running Karma server with Jasmine Unit Test.

1. The configuration files should be included with Karma.conf.js files above.
2. The node interpreter should use the system files provided by Node.js
3. The karma package file should use system file provided by Karma



4. Once the setup is done and pressed “ok”. The PhpStorm shall allow developer to run the automated test with its integrated terminal.



3. Test Documentation Requirements

In this section, the functional requirements of the calculator web application are defined. Moreover, it specifies system's features that are tested and the features that are not covered during test phase. Besides that, at the end of the section, certain criterias are defined in order to decide whether the features pass or fail the test.

3.1 System Requirements

The section provides the functional requirements that describe the functionality of the calculator web application and the non functional requirements which specifies the application characteristics.

3.1.1 Functional Requirements

The functional requirements are organized based on the functional hierarchy of the calculator web application including data input, process and result output. Each functional requirement is associated with an unique ID and a label to be referenced in other part of the document as well as a title and a brief description of the addressed function.

3.1.1.1 ***Data inputs with number entry (Inputs)***

Label	ID	Title	Description
3.1.1.1.1	**FR1**	Input Data Type	The application should only accept numbers in form of integers or decimal values.
3.1.1.1.2	**FR2**	Negation of the Input Number	The application should provide the user with the ability to negate the currently entered number by clicking on the negation button.
3.1.1.1.3	**FR3**	Length of the Input Number	If any operation is not entered, the application should continue accepting the input digits from

			the user until the input number reaches the length of 10 digits.
3.1.1.1.4	**FR4**	Input Termination of the First Operand	The application shall terminate the input of the current number when the user clicks on any of the arithmetic operatoror buttons and it shall consider the input value as the first operand.
3.1.1.1.5	**FR5**	Input Termination of the Second Operand	After the first operand is entered, the application shall terminate the input of the current number when the equal button is clicked and it shall consider the input value as the second operand.
3.1.1.1.6	**FR6**	Clear Current Values	The application should provide the user with ability to remove all the existence value in display and promote the user to enter a new arithmetic operation.

3.1.1.2 ***Arithmetic Operation (Processes)***

Label	ID	Title	Description
3.1.1.2.1	**FR7**	Type of Arithmetic Operations	The application should only perform the 4 main arithmetic operations which are subtraction, addition, multiplication and division
3.1.1.2.2	**FR8**	Calculation Process	The application should calculate the first operand and second operand based on the arithmetic operation entered and provide output based on computation.
3.1.1.2.3	**FR9**	Division on Zero	The application should reject division on Zero and show an error message.

3.1.1.3 ***Output Values (Outputs)***

Label	ID	Title	Description
3.1.1.3.1	**FR10**	Input Value Display	The application shall display the currently entered number on the screen.
3.1.1.3.2	**FR11**	Operator Display	The application shall display the operator on the screen when the user clicks on one of the relevant buttons.
3.1.1.3.3	**FR12**	Result Display	The application should display the correct result of the entered arithmetic operation on the screen when the user clicks the equal button.

3.1.2 Non-Functional Requirements

This section provides the necessary requirements to implement the required quality factors. Each non functional requirement is associated with an unique ID and a label to be referenced in other part of the document as well as a brief description. Each non-functional requirement is associated with an unique ID and a label to be referenced in other part of the document as well as a brief description..

3.1.2.1 Usability

Label	ID	Description
3.1.2.1.1	**NR1**	The Interface elements (e.g. labels of the buttons) should be easy to understand.
3.1.2.1.2	**NR2**	The Standard Calculator App should be easy to learn.
3.1.2.1.3	**NR3**	Error messages should clearly explain how to recover from the error.
3.1.2.1.4	**NR4**	The Standard Calculator App should meet specific user needs.
3.1.2.1.5	**NR5**	The screen layout and colour should be appealing.

3.1.2.2 Correctness

Label	ID	Description
3.1.2.2.1	**NR6**	The calculation output has to be correct according to mathematic operation .
3.1.2.2.2	**NR7**	The number entered should be display correctly on the screen. Automation Test Automation Test
3.1.2.2.3	**NR8**	The first number entered should be remembered by the system after arithmetic operation is entered.

3.1.2.2.4	**NR9**	The button should perform its programmed functionality.

3.1.2.3 Reliability

Label	**ID**	**Description**
3.1.2.3.1	**NR10**	The Standard Calculator application's mean time between failures shall be at least 1 month.
3.1.2.3.2	**NR11**	The probability that the Standard Calculator application fails to process user request shall not exceed 3% per year.

3.1.2.4 Maintainability

Label	**ID**	**Description**
3.1.2.4.1	**NR12**	The time period from finding a critical bug until it is fixed should not be more than 1 week. A non-critical error can be fixed within 2 weeks after the bug is found.
3.1.2.4.2	**NR13**	The code should adhere to specific standards in order to find bugs with minimal time and effort.

3.1.2.5 Flexibility

Label	**ID**	**Description**
3.1.2.5.1	**NR14**	The Standard Calculator application should be easy to adapt future changes and upgrades. The code should be written in a way that it allows implementation of new functions.

3.1.2.5.2	**NR15**	The Standard Calculator application should adapt different user levels and needs.

3.1.2.6 Testability

Label	**ID**	**Description**
3.1.2.6.1	**NR16**	Test environments should be built for the application to allow testing of the application's different functions.

3.1.2.7 Portability

Label	**ID**	**Description**
3.1.2.7.1	**NR17**	The Standard Calculator application can run on any web-enabled device.

3.2 Features to be tested

This subsection identifies all system features and functions that are to be tested. For each features listed in this section a reference to the associated system requirement is also provided.

- **Correctness of the input value**

The numbers entered by the user are tested in order to ensure that the calculator is performing the arithmetic operations on user's desired values. Testing of this feature is specifying the correctness requirements with IDs NR7 and NR8.

- **Negation of the number**

The negation feature is covered by the test in order to check whether the value of the number is negated when the user clicks the negation button "(-)" . The currently entered number should be negated despite of whether it is an integer or a decimal number. The functional requirement with ID FR2 is associated with this test.

- **Clear of the input values**

The clear feature allows the user to clear all the values that have been entered. The feature is tested to ensure that all variables are reset to their default values when the user clicks on the clear button "C". This test covers the requirement with ID FR6.

- **Correctness of the main four arithmetic operations results**

The computation of the four arithmetic operations supported by the calculator is tested in order to ensure that the output result is the correct result expected by the user. This feature is described in the requirement with ID NR6.

- **The limited length of the input number**

The test of this feature is performed to verify that the calculator is only accepting numbers with maximum length of 10 digits. The description of this feature is provided under the requirement FR3.

- **Operands terminator**

This feature is tested to ensure that the input of the first operand is terminated when the user clicks one of the operator buttons as well as to ensure that the input of the second operand is terminated when the user clicks "=" button. This test covers the requirements with ID FR6.

- **Rejection of zero division**

The test is carried out in order to make sure that the calculator handles zero division correctly by displaying an error message to the user. Testing of this feature is associated with requirements FR9 and NR3.

- **Portability**

This feature is tested by running the calculator application on different web browsers in order to ensure that the calculator is able to conform to the constraints imposed by these browsers. This test covers the requirement NR17.

- **Maintainability**

The maintainability testing shall test the effort required to analyze, changes and test application. This test covers the requirements NR14 and NR13.

3.3 Features not to be tested

This section identifies application's features that are not covered by the test and the reason behind excluding these features.

- **Flexibility**

The flexibility factor is not included in the test process since the developed application is simple calculator web application. It assumed that the application will be adapted by different user levels with minimum mathematical knowledge.

3.4 Pass/Fail criteria

This section specifies the criteria to be used to decide whether the features have passed or failed the test.

3.4.1 Unit Test pass/fail criteria

The feature passes the unit test if it is implemented correctly and it conforms to the specification. This includes automation unit test and maintainability test. If the tested feature does not meet the specification, it will fail the test and the issue will be reported.

3.4.2 Integration Test pass/fail criteria

Application modules and features are grouped together and tested to ensure that they conforms to the specification and the interfacing between the modules does not affect their functionalities. This includes recovery test and portability test. If the integration between the modules results in a failure to meet the specification, it will fail the test and the issue will be reported.

3.4.3 System Test pass/fail criteria

System test is performed when all software modules are implemented in order to ensure that all the modules meets the functional requirements of the calculator web application. This includes compliance and conformance test. If any module fails to meet the functional requirements, it will fail the test and the issue will be reported.

3.4.2 Acceptance Test pass/fail criteria

Acceptance test is performed in order to ensure that the calculator web application is stable. This includes usability test and documentation test. If the system is found to be unstable, it will fail the test and the issue will be reported.

4. Test plan

4.1 Unit Test

Unit testing is a software testing method by which individual "units" of source code, sets of one or more computer program functions are tested to determine whether they are implemented correctly. Unit test answer the question: **"Does this piece of code work exactly the same as it last time I ran this test?"** It is a lowest level of software testing where individual function and lines of a software are tested. The purpose is to validate that behavior and action of the tested software function performs as expected. This is normally performed by software developers.

4.1.1 Automation Unit Testing

Automation unit testing is a functional testing helps identify software defects easily with the help of Karma and Jasmine integration with PhpStorm. We can run a periodic check on units behavior to ensure the modules is not break during the development process. Test-driven development is applied here to generate test case and production codes at the same time to ensure good quality of software. The test case are as follow:

Test Case ID: AUT1

Requirement Covered: FR1 FR6 NR9

Test Case Description : The clear button should return all value to zero.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	125361	125361	125361	**Pass**
2	Clear button pressed	0	0	**Pass**

Test Case ID: AUT2

Requirement Covered: FR1 FR4 FR5 FR7 FR8 FR10 FR11 FR12

Test Case Description : The calculator should perform the addition operation correctly.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	2315	2315	2315	**Pass**
2	+	+	+	**Pass**
3	4526	4526	4526	**Pass**
4	=	= 6841	= 6841	**Pass**

Test Case ID: AUT3

Requirement Covered: FR1 FR4 FR5 FR7 FR8 FR10 FR11 FR12

Test Case Description : The calculator should perform the subtraction operation correctly.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	5862	5862	5862	**Pass**
2	-	-	-	**Pass**
3	2356	2356	2356	**Pass**
4	=	3506	3506	**Pass**

Test Case ID: AUT4

Requirement Covered: FR1 FR4 FR5 FR7 FR8 FR10 FR11 FR12

Test Case Description : The calculator should perform the division operation correctly.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	4123	4123	4123	**Pass**
2	/	/	/	**Pass**
3	8	8	8	**Pass**

4	=	515.375	515.375	**Pass**

Test Case ID: AUT5

Requirement Covered: FR1 FR4 FR5 FR7 FR8 FR10 FR11 FR12

Test Case Description : The calculator should perform the multiplication operation correctly.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	896541	896541	896541	**Pass**
2	*	*	*	**Pass**
3	63	63	63	**Pass**
4	=	56482083	56482083	**Pass**

Test Case ID: AUT6

Requirement Covered: FR1

Test Case Description : The calculator shall display number or decimal based on input.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	12	12	12	**Pass**
2	.	.	.	**Pass**
3	123	12.123	12.123	**Pass**

Test Case ID: AUT7

Requirement Covered: FR1

Test Case Description : The calculator must reject any division on Zero and the output should be "undefined value".

Steps	Test Data	Expected Output	Actual Output	Test Status
1	8	8	8	**Pass**
2	/	/	/	**Pass**
3	0	0	0	**Pass**

4	=	Notify Message and undefined value appears	Notify Message and undefined value appears	**Pass**

Test Case ID: AUT8

Requirement Covered: FR1 FR3

Test Case Description : The calculator only accept a maximum of 10 numbers or decimal as input.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	1234567891	1234567891	1234567891	**Pass**
2	Press any number	1234567891	1234567891	**Pass**

Test Case ID: AUT9

Requirement Covered: FR1 FR7

Test Case Description : The arithmetic operator must be displayed on the screen.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	8	8	8	**Pass**
2	+	+	+	**Pass**

Test Case ID: AUT10

Requirement Covered: FR1 FR2

Test Case Description : The calculator shall convert values into negative once negative (-) button is pressed

Steps	Test Data	Expected Output	Actual Output	Test Status
1	8955	8955	8955	**Pass**
2	-	-8955	-8955	**Pass**

4.1.2 Maintainability Testing

The maintainability testing shall test the effort required to analyze, changes and test application. The **Corrective Maintenance** time taken to diagnose and fix problems identified within the system should be measure. **Perfective Maintenance** are tested to obtain the effort required to enhance the system. Last but not least, **preventive maintenance** is taken to reduce future maintenance cost.

Test Case ID: MT1

Requirement Covered: NR12 NR13 NR16

Test Case Description : The program records the duration to find defects in system. (Corrective Maintenance Testing).

Steps	Test Procedure	Expected Duration for debugging	Actual Duration for debugging	Test Status
1	Modify a line of codes			**Pass**
2	Run Karma Test	5s (Error obtained in specific lines)	5s (Error obtained in specific lines)	**Pass**

Test Case ID: MT2

Requirement Covered: NR12 NR13 NR14 NR16

Test Case Description : The program records the duration to enhance functionality of system. (Perfective Maintenance Testing).

Steps	Test Procedure	Expected Duration for debugging	Actual Duration for debugging	Test Status
1	Add multiply function into system.	5mins	3mins	**Pass**
2	Pass Karma Test	30s	5s	**Pass**
3	Add subtraction function into system.	5mins	2mins	**Pass**
4	Pass Karma Test	30s	10mins	**Fail**

Test Case ID: MT3

Requirement Covered: NR12 NR13 NR14 NR16

Test Case Description : The program records the duration to debug implemented functionality in system (Preventive Maintenance Testing).

Steps	Test Procedure	Expected Duration for debugging	Actual Duration for debugging	Test Status
1	Run Karma Test	5s	5s	**Pass**
2	Debug subtraction function in system	10mins	1min	**Pass**
3	Pass Karma Test	30s	10s	**Pass**

4.2 Integration Test

Integration test is the phase in software testing in each individual software modules and units are combined and tested as a group. During the process, the functionality of each modules **should not be affected** by other functions after the integration and works well with other component of the software.

4.2.1 Recovery Testing

Recovery testing is a non-functional testing technique performed in order to determine how quickly the calculator application **back-end server connection** can recover after it has gone through **browser crash**. Recovery testing is to verify the time taken and possibilities of recovery successful after the system is not properly performed.

Test Case ID: RT1

Requirement Covered: NR10 NR11

Test Case Description : The program should be recovered if back-end server connection crashed unexpectedly.

Steps	Test Procedure	Expected actions	Actual actions	Test Status
1	Establish web server connection with Karma Server.	Karma server establish connection on port 9876.	Karma server establish connection on port 9876.	**Pass**
2	Close the connection and make the browser crash	Re-establish connection	Re-establish connection	**Pass**

4.2.2 Portability Testing

Portability testing is a non-functional testing technique to determining the degree of which the software component or the application can be effectively and efficiently **running in different browser** environments.

Test Case ID: PT1

Requirement Covered: NR17

Test Case Description : The program should establish connection and run on Firefox browser properly.

Steps	Test Procedure	Expected actions	Actual actions	Test Status
1	Establish web server connection with Karma Server.	Karma server establish connection on port 9876.	Karma server establish connection on port 9876.	**Pass**
2	Launch Firefox browser for connection initialization.	Connection run and monitor firefox browser	Connection run and monitor firefox browser	**Pass**
3	Open calculator application on Firefox browser	Calculator perform well in Firefox	Calculator perform well in Firefox	**Pass**

Test Case ID: PT2

Requirement Covered: NR17

Test Case Description : The program should establish connection and run on Opera browser properly.

Steps	Test Procedure	Expected actions	Actual actions	Test Status
1	Establish web server connection with Karma Server.	Karma server establish connection on port 9876.	Karma server establish connection on port 9876.	**Pass**
2	Launch Opera browser for connection initialization.	Connection run and monitor Opera browser	Connection does not run on Opera browser	**Fail**
3	Open calculator	Calculator perform	Calculator perform well	**Pass**

	application on Firefox browser	well in Firefox	in Firefox	

4.3 System Test

System testing is level of software testing where the complete and integrated software is tested. This phase of testing should require no knowledge of the inner design of the codes or logic. The testing is performed on the entire system based on functional requirements stated in section 3.1.1.

4.3.1 Compliance and Conformance Testing

The compliance and conformance testing is a non-functional testing techniques to identify whether system complies with requirements of specifications and conditions, regulations and standards along with documentation. The scope of specifications and specification objectives are required to be tested with system to check consistency, completeness and correctness. The automation test case are run to obtained the coverage and gain confidence level.

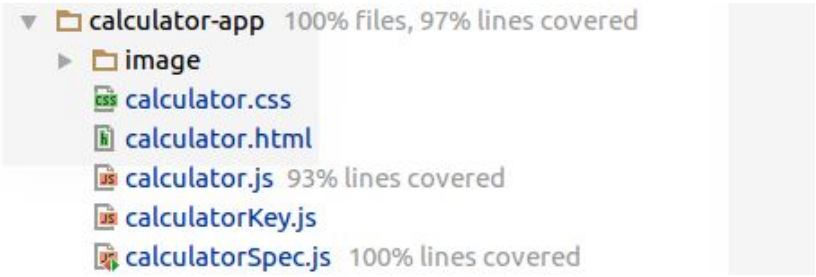


Figure 5: Code coverage for calculator application

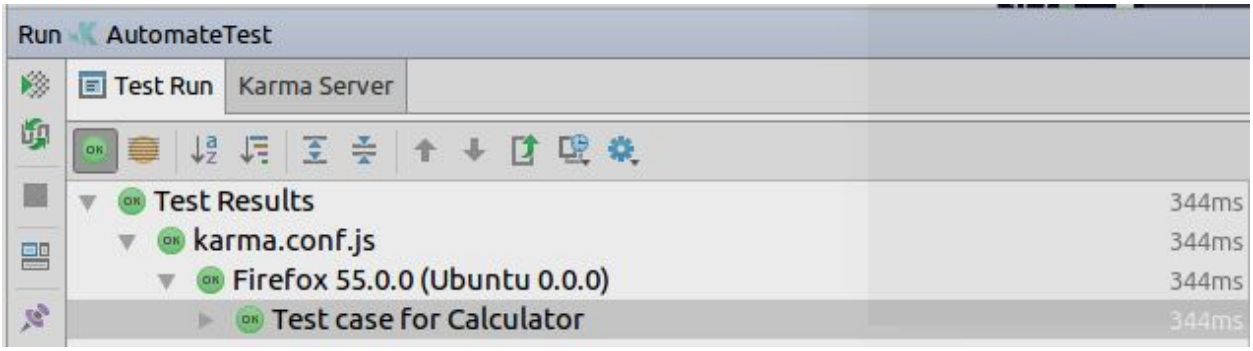


Figure 6: Execute all the test case in programs.

Figure 7: Result of all the test case execution

Test Case ID: CCT1

Requirement Covered: ALL FR (FR1-FR12)

Test Case Description : The test should ensure the system is stable, conform to requirement and consistent on providing basic functionality stated in requirement document.

Steps	Test Procedure	Expected actions	Actual actions	Test Status
1	Check conformance, consistency and test coverage of whole program	The coverage should exceed 90%	The coverage is 97%	**Pass**

Test Case ID: CCT2

Requirement Covered: ALL FR (FR1-FR12)

Test Case Description : The system modules should not break after the integration is done.

Steps	Test Procedure	Expected actions	Actual actions	Test Status
1	The functions and modules should works well after integration	The test shall not failed.	The test does not failed.	**Pass**

4.4 Acceptance Testing

Acceptance testing is a level of software testing where system is tested for acceptability from client, stakeholder or third party organization. There are two common types of acceptance testing which are alpha testing and beta testing. As the application of this project is not shown to customer or deployed, we will conduct alpha test as acceptance test to check whether calculator application are nearly full stable. The alpha test is normally performed by internal employee such as test engineers, QA engineer and "friends or family".

4.4.1 Usability Testing

Usability testing is a non-functional test techniques how user asked to complete given input or specific task on the system observed by QA or tester engineer. The user interface and user experience are recorded to test the learning curve and learnability of our calculator apps.

Test Case ID: UT1

Requirement Covered: FR2 NR1 NR2 NR4 NR5 NR7 NR9 NR15

Test Case Description: The value of the currently entered number should be negated when the user clicks the negation button regardless of whether he/she clicks the the button before, during or after entering the number.

Steps	**Test Data**	**Expected Output**	**Actual Output**	**Test Status**
1	2	2	2	**Pass**
2	(-)	-2	-2	**Pass**
3	5	-25	-25	**Pass**

Test Case ID: UT2

Requirement Covered: FR1 FR3 FR10 NR1 NR2 NR3 NR4 NR5 NR9

Test Case Description: The user is allowed to input a number with a maximum length of 10 digits and the currently entered number should be displayed on the screen.

Steps	**Test Data**	**Expected Output**	**Actual Output**	**Test Status**
1	1	1	1	**Pass**
2	2	12	12	**Pass**
3	3	123	123	**Pass**
4	4	1234	1234	**Pass**
5	5	12345	12345	**Pass**
6	6	123456	123456	**Pass**
7	7	1234567	1234567	**Pass**
8	8	12345678	12345678	**Pass**
9	9	123456789	123456789	**Pass**
10	0	1234567890	1234567890	**Pass**
11	1	1234567890	1234567890	**Pass**

Test Case ID: UT3

Requirement Covered: FR1 FR3 FR4 FR5 FR6 FR7 FR8 FR10 FR11 FR12 NR1 NR2 NR4 NR5 NR6 NR7 NR8 NR9

Test Case Description: The user is able to remove all the existence value on the screen in order to perform a new arithmetic operation.

Steps	**Test Data**	**Expected Output**	**Actual Output**	**Test Status**
1	2	2	2	**Pass**
2	+	+	+	**Pass**
3	3	3	3	**Pass**
4	=	= 5	= 5	**Pass**
5	Click “C”	0	0	**Pass**

Test Case ID: UT4

Requirement Covered: FR1 FR4 FR5 FR7 FR8 FR9 FR10 FR11 FR12 NR1 NR2 NR3 NR4 NR5 NR8 NR9

Test Case Description: The user should be notified of undefined operation when he/she tries to perform a division on zero.

Steps	**Test Data**	**Expected Output**	**Actual Output**	**Test Status**
1	5	5	5	**Pass**
2	/	/	/	**Pass**
3	0	0	0	**Pass**
4	=	undefined value	undefined value	**Pass**

Test Case ID: UT5

Requirement Covered: FR1 FR3 FR10 NR1 NR2 NR4 NR5 NR7 NR9

Test Case Description: The input number entered by the user should be displayed on the screen including numbers with a decimal value.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	2	2	2	Pass
2	.	2.	2.	Pass
3	5	2.5	2.5	Pass

Test Case ID: UT6

Requirement Covered: FR1 FR4 FR7 FR10 FR11 NR1 NR2 NR4 NR5 NR7 NR9

Test Case Description: After the first operand is entered, the sign of the arithmetic operator the user clicks should be displayed on the screen.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	3	3	3	Pass
2	+	+	+	Pass

Test Case ID: UT7

Requirement Covered: FR1 FR3 FR4 FR5 FR7 FR8 FR10 FR11 FR12 NR1 NR2 NR4 NR5 NR6 NR7 NR9

Test Case Description: After the user enters the first operand, the operator and the second operand, the application should display the correct result on the screen.

Steps	Test Data	Expected Output	Actual Output	Test Status
1	2	2	2	Pass
2	*	*	*	Pass
3	4	4	4	Pass
4	=	= 8	= 8	Pass

Test Case ID: UT8

Requirement Covered: FR1 FR3 FR4 FR5 FR7 FR8 FR10 FR11 FR12 NR1 NR2 NR4 NR5 NR6 NR7 NR8 NR9

Test Case Description: After the user enters the first operand, the operator and the second operand, the application should display the correct result on the screen.

Steps	**Test Data**	**Expected Output**	**Actual Output**	**Test Status**
1	9000000	9000000	9000000	**Pass**
2	*	*	*	**Pass**
3	2000000	2000000	2000000	**Pass**
4	=	= 18000000000000	= 1800000000000	**Fail**

Test Case ID: UT9

Requirement Covered: FR1 FR3 FR4 FR5 FR7 FR8 FR10 FR11 FR12 NR1 NR2 NR4 NR5

Test Case Description: After performing an arithmetic operation, the user can enter a new operand for the next arithmetic operation. The input should be displayed on the screen regardless of the currently displayed value.

Steps	**Test Data**	**Expected Output**	**Actual Output**	**Test Status**
1	50000	500000	500000	**Pass**
2	*	*	*	**Pass**
3	200000	200000	200000	**Pass**
4	=	= 10000000000	= 10000000000	**Pass**
5	1	1	= 10000000000	**Fail**

4.4.2 Documentation Testing

Documentation testing is a non-functional test techniques to ensure activities of every phase in software development life cycle are conform with requirement specifications. The test case, plan and procedure are documented according to IEEE Documentation standard to reflect the quality of calculator applications. The implicit and explicit requirement are stated in document help estimating testing effort required, test coverage and requirement tracking for the software team. The traceability matrix is attached to satisfy the documentation testing in section 5.

Section 5: Traceability matrix

		Functional Requirement												Non-functional Requirements																				
Reqs. ID	Reqs. Tested	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FR12	NR1	NR2	NR3	NR4	NR5	NR6	NR7	NR8	NR9	NR10	NR11	NR12	NR13	NR14	NR15	NR16	NR17				
Test cases		19	4	9	11	10	4	12	11	3	13	11	10	8	8	1	8	8	3	6	2	7	1	1	3	3	2	1	3	2				
AUT1	3	x				x																												
AUT2	8	x		x	x		x	x	x	x	x	x																						
AUT3	8	x		x	x		x	x	x	x	x	x																						
AUT4	8	x		x	x		x	x	x	x	x	x																						
AUT5	8	x		x	x		x	x	x	x	x	x																						
AUT6	1	x																																
AUT7	1	x																																
AUT8	2	x		x																														
AUT9	2	x					x																											
AUT10	2	x	x																															
MT1	3																							x	x			x						
MT2	4																							x	x	x		x						
MT3	4																						x	x	x	x		x						
RT1	2																					x	x											
PT1	1																													x				
PT2	1																													x				
CCT1	11	x	x	x	x	x	x	x	x	x	x	x																						
CCT2	11	x	x	x	x	x	x	x	x	x	x	x																						
UT1	8		x										x	x		x	x		x		x						x							
UT2	9	x		x						x			x	x	x	x	x				x													
UT3	17	x		x	x	x	x	x		x	x	x	x	x		x	x	x	x	x														
UT4	18	x		x	x	x	x	x		x	x	x	x	x		x	x	x	x	x														
UT5	9	x		x						x			x	x		x	x		x		x													
UT6	11	x			x		x			x	x	x	x	x		x	x		x		x													
UT7	16	x		x	x		x	x		x	x	x	x	x		x	x	x	x		x													
UT8	17	x		x	x		x	x		x	x	x	x	x		x	x	x	x	x	x													
UT9	13	x		x	x		x	x		x	x	x	x	x		x	x																	

6. Results Discussion

6.1 Automation Test

Ultimately, we have found that automation unit test covered all the functionality of modules according to system requirements. Test-driven development reduces the effort on testing in the testing phase and increases maintainability and testability of the program. Even though the developer required to spend extra effort to write test cases and production codes during the development phase, the approach helps write clean and useful system. There are no redundant codes and test cases found in the entire program. According to traceability matrix, the automation test also covered all the functional test of the system to ensure every single unit and function behaviour perform correctly.

6.2 Maintainability Test

Jasmine Unit Test with Karma spawns a new server and compiler to check the syntax and behaviour of JavaScript. This method helps developer easier to identify defects and problem compare to traditional development. According to results obtained from corrective maintenance testing, the software team require running the karma test to determine the problem and the error will be shown in particular lines along with the test case message. Software team can read the automation test case written previously by the programmer who responsible to develop the function to identify or learn the behaviour immediately. The effort on maintaining the system is reduced with separation and concern and modularity.

Last but not least, perfective maintenance testing shows duration to enhance the functionality of system requires the longer duration compared to expected output. This is because a developer requires to concern and pass all the test cases written previously by the team while adding new lines or function into the system. A developer might need extra effort

on refactoring the codes to pass the test by altering the internal structure without affecting the external behaviour.

6.3 Recovery Test

Base on the results obtained, the system back-end connection can recover in a particular time once the browser is broken. Since the application is not deployed, the testing attempt to host the application in localhost and monitored with karma server to ensure the application is working well. As the tester try to close and crash the calculator application, the software has successfully reestablished the connection and running again.

6.4 Portability Test

The application user interface is able to run on multiple browsers and the arrangement of buttons and components is still compatible with Firefox and Opera. However, the system connection is not established on Opera browser due to missing plugins.

6.5 Compliance and conformance testing.

The testing is conducted when the system is fully integrated, it is found that overall program and function possess test coverage of 97% which exceed satisfaction level. We had gain confidence that system are performing in proper manner which conform to requirement specification after the integration. The functionality of system also meet system requirements and does not affect other modules during the integration of system had been done.

6.6 Usability Testing

The system is tested on front-end without knowing the logics of codes, alpha beta tester will test the system based on given steps or instruction given. We had observed that user are

familiar and easy to use our system. This is because calculator application does not require a steep learning curve to understand the system operation. Moreover, we had also discovered several defects of our system based on alpha beta user input. There are several non-functional requirements are not implemented well and discovered by an internal user.

7. Critical Comments and Conclusion

According to results and reading obtain from traceability matrix, an agile testing method with test-driven development software methodology covered all the testing on the implicit and explicit requirement from start to end phase. There are no redundant test cases written and slightly reduce testing cost, effort and procedure to be taken to verify the system. The production codes written possess meaningful functionality, and there is no unused component and variable throughout the whole program.

In each level of test, we had found that automation tools do not guarantee the reliability and quality metrics of software. Similar test cases are used to exhaust the system to find bugs during the unit test and acceptance test, the alpha-beta testing can unveil potential errors and faults cannot be discovered by the unit test. This is because the unit test will only be taking cares of logic and behaviour of a system but does not address on testing user interface and user experience of the application.

In conclusion, the SDLC, SDTC and agile software methodology used in this software had successfully achieved and satisfy majority of quality metrics, functional requirements and nonfunctional requirements in the system. It can identify defects and problem earlier in every single development phase, fix earlier and respond to changes according to requirements and standards.