SE 3XA3: Test Plan

Group 11, In2gr8
Anthony Guirguis 400014834
Hoda Mohammadi 4001407527
Mikolaj Hrycko 400018523

October 28, 2017

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
October 27, 2017	0	None

1 General Information

1.1 Purpose

The purpose of the test plan is to ensure that all functional and non functional requirements of the integrate website are implemented correctly as intended in the requirement documentation. The test plan for the website is used to determine the possible tests that would be performed on the system without specifically describing each test case.

1.2 Scope

The test plan will cover all the mathematical functions that have been implemented and parsing of the equations, as well as the website interface.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
In2gr8	Integrate(Program Name)
SRS	Software Requirements Specification
PoC	Proof of Concept
JS	JavaScript
HTML	Hypertext Markup Language
OS	Operating System
UI	User Interface

1.4 Overview of Document

The test plan is used to document the testing process and the specific test cases and to make sure the testing is done correctly and efficiently.

Table 3: **Table of Definitions**

Term	**Definition**
Structural Testing	Testing derived from the internal structure of the software.
Functional Testing	Testing derived from a description of how the program functions.
Dynamic Testing	Testing which includes having test cases run during execution.
Static Testing	Testing that does not involve program execution.
Manual Testing	Testing conducted by people.
Automated Testing	Testing that is run automatically by software.

2 Plan

2.1 Software Description

Software will provide users with an online environment within which they can input mathematical expressions and equations, that will then be solved by integration, differentiation or simple calculations. The software is made available for use through a user friendly website.

2.2 Test Team

The test team includes all three project members that are Anthony Guirguis, Hoda Mohammadi and Mikolaj Hrycko.

2.3 Automated Testing Approach

There will be test suites for every JavaScript file created, as well as an integrated test suite for all of the files. Once a file is updated, its own test suite is run for error checking, and if everything passes, then the integrated suite is checked.

2.4 Testing Tools

The tool that will be used for testing the software is Mocha which is a JS testing framework that runs on node.js.

2.5 Testing Schedule

here

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

Expression Input

1. FS-EI-1

   Type: Static, Manual

   Initial State: Website that has an input box for the expression.

   Input: A set of random characters.

   Output: The exact expression is used as the input for the parser function.

   How test will be performed: A random expression is entered in the input box and manually checked if the exact expression is sent to the parser function as it's parameter.

3.1.2 Input Calculation

Parsing

1. FS-P-1

   Type: Functional, Dynamic, Manual

   Initial State: The parser function that takes an expression as argument.

   Input: A simple expression with valid syntax

   Output: An array that stores each element of the expression at a separate index.

How test will be performed: A simple expression with valid characters (numbers, operations, brackets, etc.) is used as an argument for the parser function and the output is manually checked to make sure it is an array that has each number, operation, bracket, and variable stored in order, at a separate index.

2. FS-P-2

   Type: Functional, Dynamic, Manual

   Initial State: The parser function that takes an expression as argument.

   Input: A simple expression with invalid syntax

   Output: An error stating the input expression contains invalid characters.

   How test will be performed: A simple expression with invalid characters (such as $, &, , etc.) is used as an argument for the parser function and the output is manually checked for an error as output.

3. FS-P-3

   Type: Functional, Dynamic, Manual

   Initial State: The parser function that takes an expression as argument.

   Input: A complex expression with valid syntax

   Output: An array that stores each element of the expression at a separate index.

   How test will be performed: A complex expression with valid characters (numbers, operations, brackets, etc.) is used as an argument for the parser function and the output is manually checked to make sure it is an array that has each number, operation, bracket, and variable stored in order, at a separate index.

Integration

1. FS-I-1

   Type: Functional, Dynamic, Manual

   Initial State: The integration function that takes a parsed array as argument.

Input: An expression parsed in an array that would require a simple integration technique to be solved.

Output: The correct integration solution to the inputed expression in form of a string.

How test will be performed: An array composed of an expression's elements is passed on as an argument to the integration function and the returned string from the function is manually checked to see whether it was integrated correctly or not.

2. FS-I-2

   Type: Functional, Dynamic, Manual

   Initial State: The integration function that takes a parsed array as argument.

   Input: An expression parsed in an array that would require a complex integration technique to be solved.

   Output: The correct integration solution to the inputed expression in form of a string.

   How test will be performed: An array composed of an expression's elements is passed on as an argument to the integration function and the returned string from the function is manually checked to see whether it was integrated correctly or not.

Differentiation

1. FS-D-1

   Type: Functional, Dynamic, Manual

   Initial State: The differentiation function that takes a parsed array as argument.

   Input: An expression parsed in an array that would require a simple integration technique to be solved.

   Output: The correct differentiation solution to the inputed expression in form of a string.

   How test will be performed: An array composed of an expression's elements is passed on as an argument to the differentiation function

and the returned string from the function is manually checked to see whether it was differentiated correctly or not.

2. FS-D-2

   Type: Functional, Dynamic, Manual

   Initial State: The differentiation function that takes a parsed array as argument.

   Input: An expression parsed by elements in form of an array that would require a complex differentiation technique to be solved.

   Output: The correct differentiation solution to the inputed expression in form of a string.

   How test will be performed: An array composed of an expression's elements is passed on as an argument to the differentiation function and the returned string from the function is manually checked to see whether it was differentiated correctly or not.

Simple Calculation

1. FS-SC-1

   Type: Functional, Dynamic, Manual

   Initial State: The simple evaluation function that takes a parsed array as argument.

   Input: An expression parsed by elements in form of an array.

   Output: The correct solution to the inputed expression in form of a string.

   How test will be performed: An array composed of an expression's elements is passed on as an argument to the simple expression function and the returned string from the function is manually checked to see whether it was calculated correctly or not.

2. FS-SC-2

   Type: Functional, Dynamic, Manual

   Initial State: The simple evaluation function that takes a parsed array as argument.

Input: An expression parsed by elements in form of an array that is not solvable.

Output: An error stating the equation cannot be solved.

How test will be performed: An array composed of an expression's elements is passed on as an argument to the simple expression function and it is checked if an error is returned.

3.2 Tests for Nonfunctional Requirements

3.2.1 Performance

Speed performance

1. NF-SP-1

   Type: Dynamic, Manual

   Initial State: Website loaded on a browser connected to an average speed internet.

   Input/Condition: Simple expression to be parsed and differentiated

   Output/Result: The results returned within 2 seconds.

   How test will be performed: Input the expression in the input section of the website, and wait for the expression to be parsed and differentiated and returned as output. Start a timer before input and stop after output to see whether it is under 2 seconds or not.

2. NF-SP-2

   Type: Dynamic, Manual

   Initial State: Website loaded on a browser connected to an average speed internet.

   Input/Condition: Complex expression to be parsed and integrated

   Output/Result: The results returned within 2 seconds.

   How test will be performed: Input the expression in the input section of the website, and wait for the expression to be parsed and integrated and returned as output. Start a timer before input and stop after output to see whether it is under 5 seconds or not.

Precision

1. NF-P-1

   Type: Dynamic, Manual

   Initial State: Website loaded on a browser

   Input/Condition: Simple expression to be parsed and evaluated with numerical evaluation function

   Output/Result: The results returned with 4 decimal accuracy.

   How test will be performed: Input the expression in the input section of the website and check the returned output to see whether it is accurate within 4 decimal place.

3.2.2 Maintainability and Support

Adaptability

1. NF-A-1

   Type: Static, Manual

   Initial State: A collection of devices with a browser and internet access.

   Input/Condition: The website URL in the browser and input a simple expression in the input box.

   Output/Result: Website loaded correctly on the browser and the output is given.

   How test will be performed: Load the website on many different devices and different browsers to see if the UI works on all and test a simple expression calculation to check whether the functions return the correct output on any device or browser.

3.3 Traceability Between Test Cases and Requirements

4 Proof of Concept Testing

4.1 Parser

1. Type: Dynamic, Manual

Initial State: The parser function takes a mathematical expression in the form of a string as an argument.

Input: A valid mathematical expression with each term separated by a space. For example 2x + 5.

Output: An array with each term in a separate index. Array = [2, x, +, 5]

How test will be performed: A valid mathematical expression with any combination of valid mathematical expression terms along with a space in between each term shall be inputted as a string. The parser function will then look at term separated by a space and split it up whenever the type of term changes. For example with 2x + 5, the parser will look at 2x and split them up because its a number and a variable. Then it will look after the space and put + and 5 in their own indices. An array with the split up values is then returned.

2. Type: Dynamic, Manual

   Initial State: The parser function takes a mathematical expression in the form of a string as an argument.

   Input: A valid mathematical expression with each term separated by a space. For example

   $$(5x^2 + 3x + cos(2))$$

   .

   Output: An array with each term in a separate index. Array =

   $$[5, x, "", "2", +, "3", "x", "+", "cos", "(", "2",)]$$

   How test will be performed: A valid mathematical expression with any combination of valid mathematical expression terms along with a space in between each term shall be inputted as a string. The parser function will then look at term separated by a space and split it up whenever the type of term changes. This is done the same way it was done before.

4.2 Integration

1. Type: Dynamic, Manual

Initial State: The integrate function, takes a mathematical expression in the form of a string as an argument which is passed on to the parser and receives an array with the parsed data.

Input: A simple valid mathematical expression with each term separated by a space. For example 2x + 5.

Output: The output will be

$$2x^2 + 5x + C$$

How test will be performed: The test will be performed by first inputting a mathematical expression in the form of a string. After parser receives the string, it sends it over to the parser function which returns a parsed array with all the data. The integrate function will then process the data and return the final answer. With the current state of the integrate function at the time of the proof of concept, the function does not automatically get simplified yet.

2. Type: Dynamic, Manual

   Initial State: The integrate function, takes a mathematical expression in the form of a string as an argument which is passed on to the parser and receives an array with the parsed data.

   Input: A simple valid mathematical expression with each term separated by a space. For example

   $$5x^2 + 2x + 1$$

   .

   Output: The output will be

   $$(5x^3)/3 + x^2 + x + C$$

   How test will be performed: The test will be performed by first inputting a mathematical expression in the form of a string. After parser receives the string, it sends it over to the parser function which returns a parsed array with all the data. The integrate function will then process the data and return the final answer.

4.3 Numerical Evaluator

1. Type: Dynamic, Manual

   Initial State: The numerical evaluator function, takes a mathematical expression in the form of a string as an argument which is passed on to the parser and receives an array with the parsed data.

   Input: A simple valid mathematical expression with each term separated by a space. For example $(2^2)/2$.

   Output: The output will be 2.

   How test will be performed: The test will be performed by first inputting a mathematical expression in the form of a string. After parser receives the string, it sends it over to the parser function which returns a parsed array with all the data. The numerical evaluator function will then process the data and return the final answer which is 2.

2. Type: Dynamic, Manual

   Initial State: The numerical evaluator function, takes a mathematical expression in the form of a string as an argument which is passed on to the parser and receives an array with the parsed data.

   Input: A simple valid mathematical expression with each term separated by a space. For example $(4^2)/2$.

   Output: The output will be 8.

   How test will be performed: The test will be performed by first inputting a mathematical expression in the form of a string. After parser receives the string, it sends it over to the parser function which returns a parsed array with all the data. The numerical evaluator function will then process the data and return the final answer which is 8.

5 Unit Testing Plan

5.1 Types of Tests

The unit tests will be comprised of test suites for every JavaScript file and there individual functions. They will test the functions in normal, as well as boundary cases. These tests will be automated, structural and functional tests.

5.2 Drivers

All files tested will have drivers in place to do unit testing, as all of them derive values from other files. The drivers will be dummy functions that contain the inputs for the files and contain the inputs for all of the tests.

5.3 Coverage Metrics

The test cases will be critically selected as to cover most if not all lines of code. Some tests will also be testing many functions in parallel as certain inputs may need to use a significant amount of the code to produce the output. This has the benefit of testing coverage in terms of the nested complexity of the input.

5.4 Unit testing of internal functions

5.4.1 Parser

Initial State: The parser takes a mathematical expression in the form of a string as an argument.

Example Inputs Tested:

- "314159265"
- "x"
- "xˆ2 + 3x"
- "cos(x)"
- "sin((575eˆ(x+4) + x + 5)/2)/2"

5.4.2 Integration

Initial State: The integrate function, takes a mathematical expression in the form of a string as an argument which is passed on to the parser and receives an array with the parsed data.

Example Inputs Tested:

- '2718218'
- 'x'
- 'x^2', '+', '3x'
- " 'cos', '(', 'x', ')""

5.4.3 Derivation

Initial State: The derivative function, takes a mathematical expression in the form of a string as an argument which is passed on to the parser and receives an array with the parsed data.

Example Inputs Tested:

- '1615'
- 'x'
- 'x^2', '+', '3x'
- " 'cos', '(', 'x', ')""
- "'sin', '(', '5x', ')', 'x^2' "
- "'sin','(','(','575e^','(','x+4',')','+','x','+','5',')/2',')/2""

5.4.4 Simple Calculations

Initial State: The simple calculations takes a mathematical expression in the form of a string as an argument.

Example Inputs Tested:

- "13737 + 1"
- "10!"
- "(5^10)/2"
- "0/0"

- "2^0"
- "tan(45)"
- "asin(e^(pi))"
- "-2cos((pi + pi*cos(pi))/4) + 5!/120 - cos(pi)"