Random Test Data Generation

Iteration Report



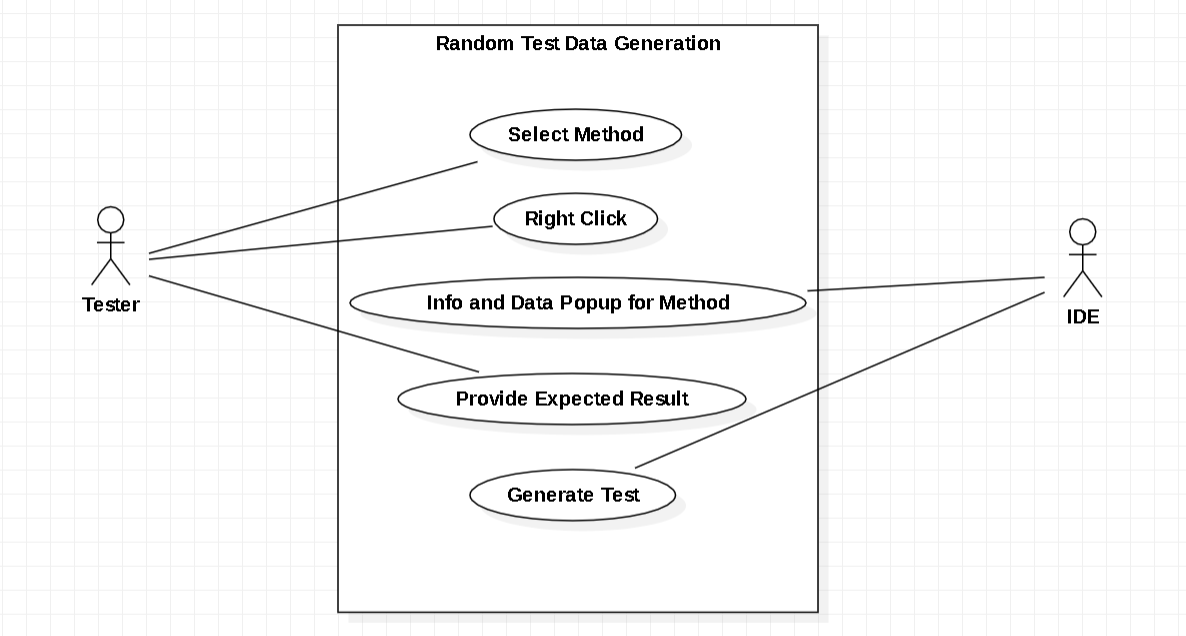
Course: Bachelor of Science (Honours) in Software Development

Name: Kyle Kinsella

Date when document was started: 06/03/2024

Supervisor: Chris Meudec

Updated use case diagram



Updated brief use cases

**Name**: Select Method

**Actor**: Tester

**Description**: In order for the tester to test a method they have to select a method that they wish to see some information about the method they have selected.

**Name**: Right Click

**Actor**: Tester

**Description**: After the tester has selected there method, they will right click. This is a very important feature as this is the core of my project.

**Name**: Info and Data Popup for Method

**Actor**: IDE

**Description**: After the tester has right click on the selected method, a popup will be shown to the screen to show the tester what method they have picked and the generated random test data.

**Name** - Provide Expected Result

**Actor** - Tester

**Description** - After the tester is shown the pop up from the last use case, they will be asked to provide the expected result depending on what generated random test data that was given to the tester initially was.

**Name**: Generate Test

**Actor**: IDE

**Description**: After all of the above use cases have been executed, a button named “Generate Test” will show up on the testers screen, when the tester clicks this button a JUnit test will automatically be made and shown to the tester.

Below are some example of JUnit tests with the “add” method.

public class iteration2 {

public static int add(int num1, int num2) {

return num1 + num2;

}

public static int sub(int num1, int num2) {

return num1 - num2;

}

}

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class testingAddMethod {

/// Test's for the add method///

@Test

public void test1() {

int num1 = 5;

int num2 = 4;

int ans = iteration2.add(num1, num2);

assertEquals(9, ans);

}

@Test

public void test2() {

int num1 = 3;

int num2 = 2;

int ans = iteration2.add(num1, num2);

assertEquals(5, ans);

}

@Test

public void test3() {

int num1 = 2;

int num2 = 6;

int ans = iteration2.add(num1, num2);

assertEquals(8, ans);

}

/// Test's for the add method///

}

Below are some example of JUnit tests with the “sub” method:

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class testingSubMethod {

/// Test's for another method called sub ///

@Test

public void test11() {

int num1 = 5;

int num2 = 3;

int ans = iteration2.sub(num1, num2);

assertEquals(2, ans);

}

@Test

public void test12() {

int num1 = 4;

int num2 = 2;

int ans = iteration2.sub(num1, num2);

assertEquals(2, ans);

}

@Test

public void test13() {

int num1 = 5;

int num2 = 5;

int ans = iteration2.sub(num1, num2);

assertEquals(0, ans);

}

/// Test's for another method called sub ///

}

**Fixing the speed of JUnit: How I did it**

During this iteration I have been focusing on how to optimise the running speed of JUnit. When I initially set up my workspace within eclipse for my project, I added some JUnit jar files to my project, but when I would go to run a test, JUnit would take a very long time to get running and eventually run my tests. Due to the runtime of JUnit I did some research to try and find out why JUnit is running so slow [1]. After I was done reading this website [1], I changed a file that the website said to change, to the name of my laptop, but nothing changed. Then I thought of an idea, my idea was to install a JUnit plugin into my eclipse. I found this website, with the plugin for JUnit [2], after I installed this plugin into my eclipse, my JUnit tests would run lightning fast compared to when I didn't have this plugin, it would run incredibly slow.

**Idea for a new feature in my software: But it backfired**

At the start of this iteration, I had a lot of work to do, but I was thinking of implementing a feature into my software. The feature was going to effectively allow the tester that is using the software to overwrite a file if it exists. Before I started to code out any functionality I did some research to see if this was going to be a good idea. I opened up Google and I searched for the following “can you overwrite

a file in java if the file exists”. After I was greeted with this popup from google [3]. I was a bit disappointed, but this was the only bit of research I had did so far, so I did more research to retrieve more information. I found this post online [4].

I found the Oracle Java API for the FileWriter class [6]. The FileWriter constructor takes two parameters the first parameter being the filename and the second parameter being a boolean value. If you have a file with your filename and false, by default by the FileWriter construtor, will overwrite that file if it exists. But if you have your filename and true, by default the file won’t be overwritten but whatever was in the file previously will still be there and any other additional information.

Here is an example to demonstrate what I am trying to explain:

public class selectText {

public static boolean isBool(boolean yes, boolean no, boolean b)

return yes && no && b;

}

}

public class testingisBoolMethod{

@Test

public void testingisBool() {

// variables initalized

boolean num1 = false;

boolean num2 = false;

boolean num3 = false;

//do something with the variables

boolean ans = rightClickMe.selectText.isBool(num1, num2, num3);

//check if the answer is correct

assertEquals(false, ans);

}

}

In my above example, I have a test for my isBool method. Now lets say I wanted to test this method again, I would go through the steps I need to do in order to see the file with the generated test or tests, but because the test has been made previously means that the file exists and I am testing this method again and this will result in the file being overwritten with the new data. Resulting in the previous test or tests being automatically overwritten.

I was anticipating on making this feature as I thought it would be a good feature to have for my project as it is a very fitting feature, but due to my research I won’t be able to implement this feature.

**References**

1. Symphonious. (2024). Why are my JUnit tests running so slow? Retrieved from: <https://www.symphonious.net/2013/12/16/why-are-my-junit-tests-running-so-slow/>
2. Eclipse Marketplace. (2024). JUnit Plugin, to fix JUnit runtime. Retrieved from: <https://marketplace.eclipse.org/content/junit-tools-4-spring>
3. Google. (2024). Can you overwrite a file in java if the file exists? Retrieved from: <https://www.google.com/search?client=firefox-b-d&q=can+you+overrite+a+file+in+java+if+the+file+exists>
4. Quora. (2024). How do I clear a file in Java before writing to it again? Retrieved from: <https://www.quora.com/How-do-I-clear-a-file-in-Java-before-writing-to-it-again>
5. Jenkov. (2024). Learning about FileWriter. Retrieved from: <https://jenkov.com/tutorials/java-io/filewriter.html>
6. Oracle. (2024). FileWriter Syntax. Retrieved from: <https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html>