Research Manual

JUnit Test Generator

I.T. Carlow

Software Engineering

Bachelor Degree (With Honours)

Sergio Alcocer Vázquez

C00132732

Table of Contents

1. Introduction.

The age of Technology is here, and with it, it is needed to create programs to give instructions to machines, in order to do what we want. To do so, we need to create a program. First of them were very simple. As the needs of those programs increases, the source code needed and the architecture of the application need to evolve.

Nowadays, we are used to use classes, patterns, and a lot of useful stuff, but all of that is as a result of the study of the Software Engineering. If we also add to this, the amount of great applications that help us create, edit, update our source code easily and with less errors (because they highlight singleton variables that you may have mistype, we will be able to create great applications.

These applications would have classes, modules, etc. But... how to check if they work properly or not? Shall we wait until we finish the whole program to test it?. The answer to this question is 'No, we shouldn't '. Why?. Easy, because if we have made mistakes while typing the code, and it doesn't work properly, It will be hard to find those errors. And, if we have around 20 classes, no one would like to start modifying the code to print the values of some variable on screen (a bad debug).

We will use a small (1 class) java example as a base to compare different ways of doing things.

Book.java (full code in Appendix I)

```
package book;
public class Book {
    private String title;
    private double price;

    public Book(String title, double price);
    public boolean equals(Object object);
    public double getPrice();
    public void setPrice(double price);
    public String getTitle();
    public void setTitle(String title);
}
/*
     Note that all method's body has been
   deleted to make it shorter.
*/
```

This is one of the simplest classes. It only have two attributes, its gets/sets methods, a constructor and a equals override.

- If this class is part of a bigger system, like a country book shop management system, and plenty of more classes, it would be difficult to trace an error to the class. That aims the programmer to use unit testing.

- Now imagine a software that only use this class (hypothetically). We have it working and we need to test it. To do so, we run the application and start creating books, setting and getting information, comparing two books, etc. If to test it, we have to introduce ten books, with its prices and its titles, and then check methods to change the price, etc. The first time, its OK, the second-one... still OK, but if we need to test it more, (because we change methods to make them work better, or any other reason, and each time we want to test the class, we have to put all the information by hand, it becomes a boring task.

2. Test Driven Development (TDD) [1]

2.1. Brief Description

Test Driven Development is a new technique that forces the developer to create tests before creating the source-code. By doing this, the advantages we get are that we are sure that our code will work, we don't have to waste time on debugging after creating our code and as a collateral consequence the code gets cleaner, at least the visible part.

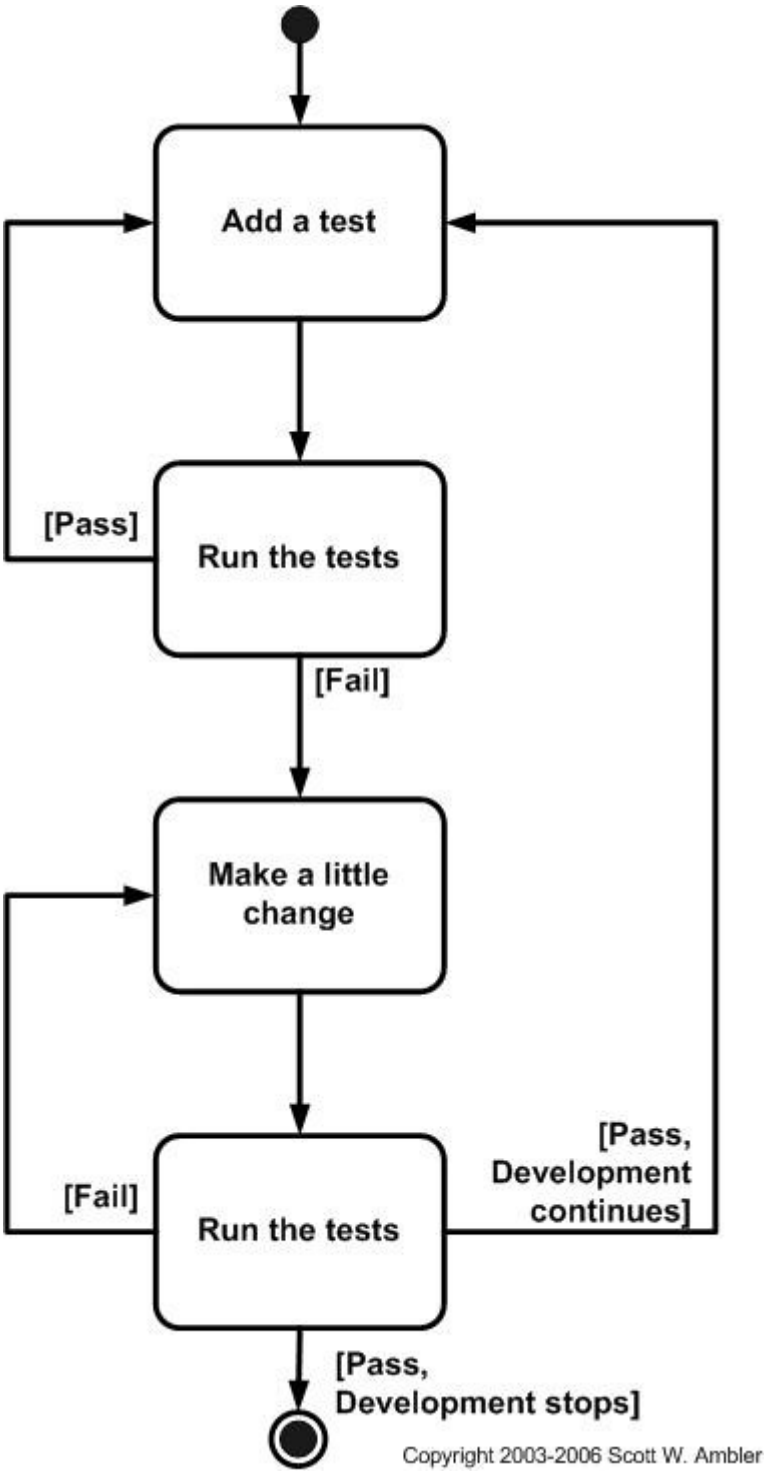
2.2. How it Works

The main idea of this method is to, firstly, create a small test with that tests some of the functionality of the class. Once you have ready the test, is the moment to start coding the class that satisfy that test. In the following figure, the sequence of steps is shown.

The basic flow is basically the following

1. Add a test (create it if empty)
2. Run the test.
3. Go to step 1 until failure
4. Write code that satisfy your tests
5. Run tests.
6. Go to step 4 until success.
7. If all functionality is developed, stop.
8. Go to step 1.

Or as a flow diagram



2.3. Advantages

One of the main advantages that this programming methodology is that, as you, as a project manager or developer, will estimate the amount of effort needed to create a working code following this method. And that estimation includes the testing and debugging sessions that, in other methodologies, are one of the problems at estimating delivery time.

Another advantage is that, as you first code the tests, you code them as it easier for you to test it. Using functions / class methods, as it is easier for you to use (as a programmer or class user). That forces you to write code in certain way that is easier to reuse, as it is mainly focused in functionality.

We will see later the advantages of this when used with XUnit.

3. XUnit & JUnit [2]

3.1. Introduction

XUnit is the word used to group all the frameworks used in different languages to support code-driven testing. There are several of them, some examples are JUnit (the one we will work with), PHPUnit, PyUnit, CUnit, etc. Using this frameworks, developers are able to create repeatable tests.

Then, back to our example, using JUnit, we would only need to create the JUnit code that tests that class and, once done, run the test. If later on, we change Book's inner code, we will only need to rerun the test and we will check if we have screw the class up or it still working.

If we increase the functionality of the Book class by adding new methods, etc. We will have to change the code of the test class, but we can keep all the previous functionality test.

That makes much easier to develop a good quality software, save time to improve the software or to use it .

In other chapter, we will show the differences between testing with or without using Junit.

3.2. Learning how to use it [3]

There are several functions related with testing in JUnit as a consequence of the inheritance of TestCase, we will try to explain them as good as we can to be able to understand the examples that are in one of the coming sections.

3.2.1 assert Functions

- assertTrue(statement) → retruns an error if statement is False
- assertFalse(statement) → returns an error if statement is True
- assertEquals(obj1, obj2) → returns an error if obj1 (object1) and obj2 are different
- assertNotSame(stat1, stat2) → returns an error if obj1 and obj2 are the same

3.2.2. setUp method

There is a method called setUp, that allows us to initialize all the common objects we are going to use in our test class. If we want to use it, we have to override it, cause it is already declared on the super-class.

3.2.3. Calling tests.

There is two ways to call tests, statically or dynamically.

When we have several tests and our tests are increasing, and adding new functions that we will have to test, the best way is to create them dynamically. (in our code we have to create the method suite() as follows)

```
public static Test suite(){
  TestSuite suite = new TestSuite();
  ... //add tests to the suite
  return suite;
}
```

3.3. Advantages

To show better how good TDD and Unit Testing are, lets list some of their advantages:

- Unit Tests are the prove you need to show that your code works.
- If while trying to improve our code, it stops working, we can always roll-back to the previous state.
- It fulfils you as a programmer when you see it actually working
- We can see that some progress is being done little by little.
- If tests codes are delivered with the class code, it helps other programmers understand how to use our class.
- It forces you to think about the functionality you want and act following that.
- It reduces costs of bugs. The bigger the code is, the more time we need to find the bug. If we know which method is the responsible of the failure, we can find it earlier.

4. Similar tools and Applications

4.1. WTTOOLS JUnit Test Generator [4]

> "Lets assume you started developing your application some weeks ago or some months ago or even some years ago. I am sure that testing your code was one of major problems. You though about automating such process. And than you heard about Unit Testing. So you would like to start use it. However how to introduce Unit Testing for large number of source files. There is too much of stupid code to write in all places again. That is where this package comes in."

This project was meant to be a first approach to a automatic JUnit Test Generator, that help developers to create JUnit tests for their developed projects, but their forum have been inactive since 2004, that makes it a little bit out-dated.

In their forums there are a huge amount of posts asking for support, and nobody has answered them, people trying to have it working, etc.

Anyway, we tried to download it (link available in Downloads section) and we weren't able to make it work neither with our example.

4.2. Test Generator [5]

This is another promising tool. It provides a easy/simple interface in which you can choose the class you want it to create the tests for.

Advantages:

- It is very simple to use.
- It is very intuitive.

Problems found:

- It doesn't even create a test for our simple Book class example.
- They are not providing support any more.
- Last release was in 2001, what makes it truly limited about the version of Java to use.
- Their web-page is not working any more
- It seams to be abandoned.

4.3. Parasoft [6]

> “Our creativity is the driving force that has allowed us to introduce so many technologies that really changed the industry. Each discovery was a triumph, but I think our greatest accomplishment is figuring out how to leverage these technologies to deliver quality as a continuous process”
>
> Adam Kolawa, Ph. D.
> Parasoft co-founder and CEO

At first sight, this looks a pretty big company, they have a lot of software that helps developers. The one we are interested on is “Parasoft Java Quality Solution” in which the assure that is able to Generate and Execute Unit Tests (“Enables the team to start verifying reliability and functionality before the complete system is ready, reducing the length and cost of downstream processes such as debugging. “) and generate automated regression tests (“Generates and executes regression test cases to detect if incremental code changes break existing functionality or impact application behavior. ”). What is the main part of our program. The advantage of our program is that is going to be freeware, and this one appears to be sharewere. And this program (Parasoft one) also includes more functionality that the user might don't want to use and don't want to purchase either.

5. The proposal

5.1. Introduction

The software produced would be used basically by two kind of users. Those who have projects with sources and with no tests, and those who haven't started yet.

5.1.1. Code analyser

This is the one for those users that already have the code written and want to add some Testing code to his/her application.

The user must specify source files, and the application would scan them to know dependencies between classes. After that, a Wizard would guide the User through all the steps to add new tests, by displaying options to choose which method would the User like to test, what parameters to use, and all the related options. Will suggest outputs (based on current output of the code), etc.

5.1.2. Wizard

This tool is for those users that are starting to create an application and want to use the TDD methodology.

A wizard will guide the user through all the steps, creating test for functions that doesn't exist, type of parameters, etc. And maybe will generate a skeleton of the class needed to fulfil the test (at least the public methods)

5.2. Differences & Advantages

The book's example is the one we are going to use to try to make clear advantages of using this technology.

5.2.1. Without TDD

If the programmer decide not to use TDD as a methodology, the software produced would be less useful, it would be easy to create (as a class), but much difficult to use from the outside. Basically, the programmer would create standard methods for each class, and some others he considers important, but the main part of the work, is going to be done outer class e.g. a class called Bookshop. This class would require certain functionality that doesn't have Book class, then, the programmer has to either modify the Book class or type the code in Bookshop class. This second option looks fine, but it carries a problem, which is that if you have to do in different classes something similar, we will have to repeat code. And that's makes us bad programmers.

Lets focus now in the problems we will have while trying to test this small amount of code. We will have to create a static main in the file to test it. This looks bad, extremely bad and if someone with an idea of JUnit, TDD, etc. puts his hands on our code, we will be considered noobs. And is not the only disadvantage, imagine that we want to reuse or lend our code, we should not give a main to test it, but we should provide a test to prove that it works fine, we are in a dead-end.

5.2.2. Testing without JUnit

We will assume now that we know the advantages of using TDD, but we don't know JUnit yet. To use TDD, we create a static main that tests the functionality of the class before start typing the class code. We start thinking on some of the functionality we need and how do we want to use it. Forcing the future class to follow that. [read part 2 for further information about this method].

All right, we have fixed the bad structure of our class, but we still have to create a static main to test it,

5.2.3. Testing with JUnit

This is the cleverest way of doing it, (at least right now). We will have the test code outside our Book's source-code, what makes it more readable, better structured.

Now, with our test code, we can decide if we want to share just our Book class or with the test classes. We don't need to 'cut' anything.

5.2.4. Using the proposed application.

The application would provide not only an easy way to create Unit Tests that even an inexperienced programmer would be able to create them, but also it would make the programmer save time.

5.2.5. Code needed to test with each methodology

For our given Book class, we will have to create the following method.

```
public static void main(String argv[]){

  Book b1 = new Book("My First Python Application",15.99);
  Book b2 = new Book("My Second Python Application",31.98);

  System.out.println(b1.getTitle() + " > " + b1.getPrice());
  System.out.println(b2.getTitle() + " > " + b2.getPrice());
  System.out.println(b1.equals(b2));
  b2.setTitle("My First Python Application");
  System.out.println("----------------");
  System.out.println(b2.getTitle() + " > " + b2.getPrice());
  System.out.println(b1.equals(b2));
  b2.setPrice(15.99);
  System.out.println("----------------");
  System.out.println(b2.getTitle() + " > " + b2.getPrice());
  System.out.println(b1.equals(b2));
}
```

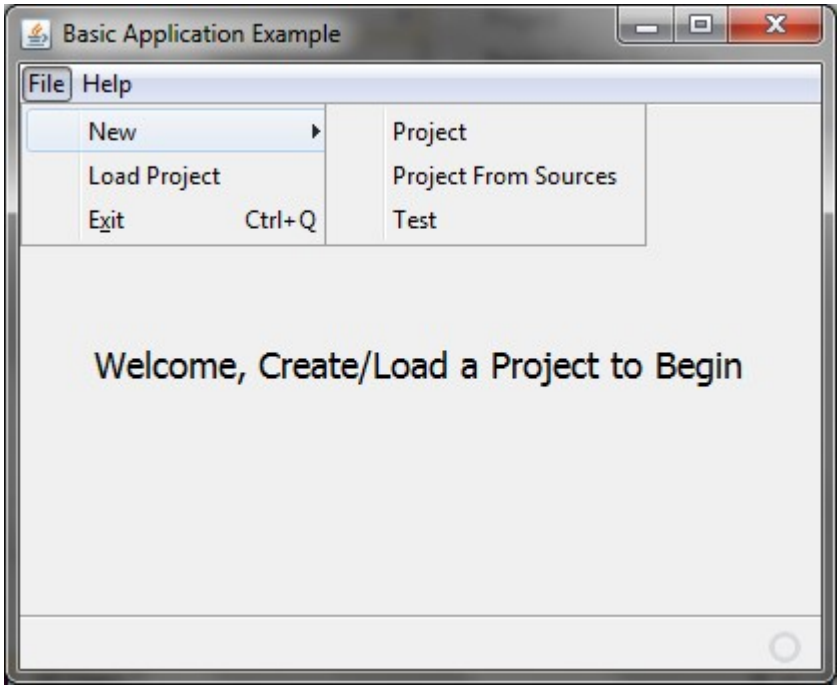
The difference between 5.2.1 y 5.2.2 is that in the first-one, this code is going to be created after creating the class, and in the other one, this code is created at first place.

As it is a simple class, there is not much difference (none) between Book classes generated with those methods.

Although with JUnit looks much more difficult (see Appendix II for an example), it is much easier to modify, and as is going to be generated, we don't have to worry about its length.

5.3 User Interfaces

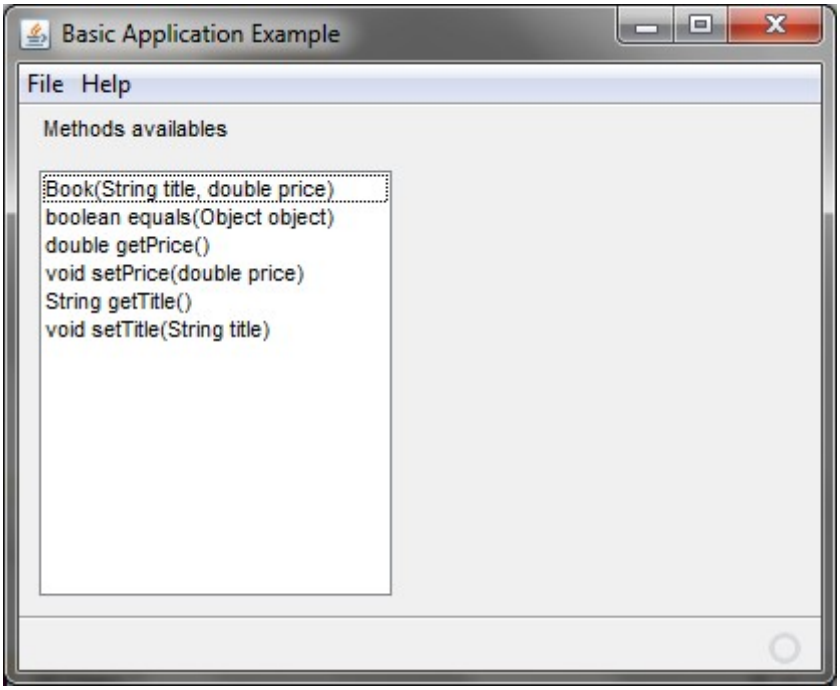
The user, would start in the following screen.



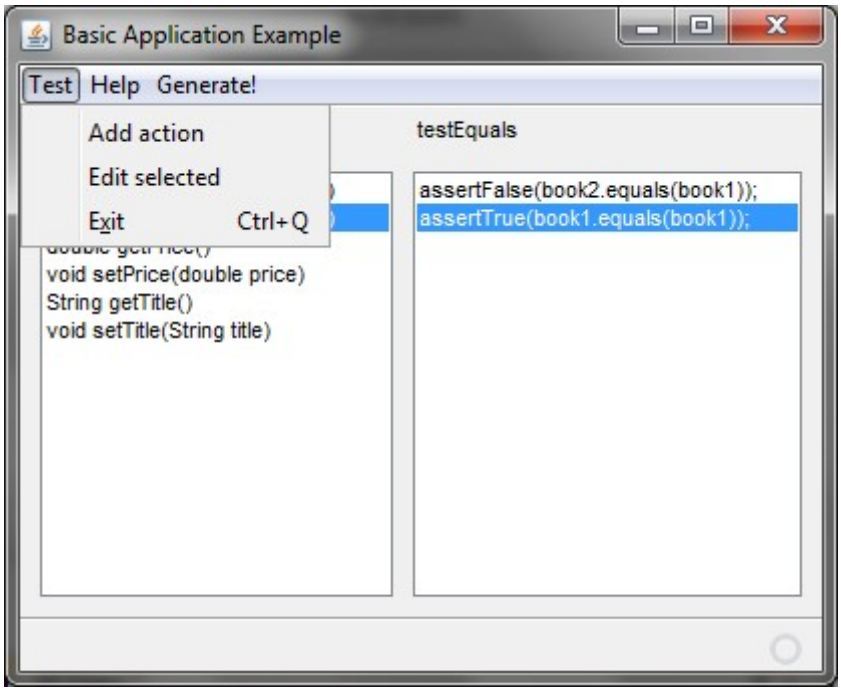
In that screen, the user can choose whether he wants to create a project / test or load an existing project.

5.3.1. Project from Sources

The user will be requested to give the location of the source file (browsing it), after that, the program will analyse the code and prompt a list of functions (public methods of the class), to start adding tests.



By choosing one of the methods shown, will display another list with the tests that are going to be performed to that method.



In the constructor is the information about the objects used in the test (in our example, book1 and book2).

And by clicking on Generate!, would make the application generate the test source-code (Appendix II).

By clicking on Add action, the user would be requested to choose an action to add to the current method. Most choices would be made automatically by the program, but will also allow the user to create a custom one.

5.3.2. New Project

The difference between this option and the previous one is that in this one, the user have to provide the prototypes of the methods that are going to be tested, and also, information about the inputs and outputs manually, mainly because there is no code that can be used to guess outputs.

The interface would be pretty much the same as the ones above, but with some slight difference. If the user clicks on Generate!, he would be asked if he also wants the skeleton of the main class to be generated. In that case, an empty class would be generated with the information provided.

5.3.3. Note about UI

The User interface shown here is a non-functional Interface, and can be modified to fit better needs in the future

6. Bibliography

[1] Introduction to Test Driven Design (TDD)
by Scott W. Ambler
http://www.agiledata.org/essays/tdd.html

[2] What is JUnit? Definition from Whatis.com
http://www.whatis.com

JUnitQuickTutorial – t2framework – A quick JUnit tutorial for beginners
by Wishnu Prasetya
http://code.google.com/p/t2framework/wiki/JUnitQuickTutorial

Twelve Benefits of Writing Unit Test First
by J. Timothy King
http://blog.jtimothyking.com/2006/07/11/twelve-benefits-of-writing-unit-tests-first

[3] Learning JUnit for Unit Testing
by Dr. Robert. L. Probert (University of Ottawa)
http://paul.rutgers.edu/~moalam/recitation5.ppt

[4] WTTOOLS JUnit Test Generator
by Artur Hefczyc
http://www.junit.org/node/20

[5] Domain for Sale
http://www.webmind.com

[6] Parasoft delivers quality as a continuous process
→ Solutions → Java
http://www.parasoft.com

7. Downloads

WWTOOLS JUnit Test Generator	http://sourceforge.net/projects/wttools/files/
Test Generator	http://sourceforge.net/projects/junittestmaker/files/Testing/1.1/TestGenerator1a.zip/download

Appendix I

```
/*
      For javaDoc commented code,
    please check the source code.
*/
package book;

public class Book {

    private String title;
    private double price;

    public Book(String title, double price){
        this.title = title;
        this.price = price;
    }

    public boolean equals(Object object){
        if (object instanceof Book){
            Book book = (Book) object;
            return  getTitle().equals(book.getTitle()) &&
                    getPrice() == book.getPrice();
        }
        return false;
    }

    public double getPrice(){
        return price;
    }

    public void setPrice(double price){
        this.price = price;
    }

    public String getTitle(){
        return title;
    }

    public void setTitle(String title){
        this.title = title;
    }
}
```

Appendix II

```
package test.book;

import book.Book;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class BookTest extends TestCase{
      private Book book1;
      private Book book2;

      public BookTest(String name) {
        super(name);
       }

      protected void setUp() throws Exception{
        super.setUp();
        book1 = new Book("My First Python Application",
                         15.99);
        book2 = new Book("My Second Python Application",
                         31.98);
       }

    protected void tearDown() throws Exception {
        super.tearDown();
        book1 = null;
        book2 = null;
    }

    public void testEquals(){
        assertFalse(book2.equals(book1));
        assertTrue(book1.equals(book1));
    }

    public void testGetPrice(){
      assertTrue(book1.getPrice() == 12.99);
      assertFalse(book1.getPrice() == 11.99);
    }

    public static Test suite(){
        TestSuite suite = new TestSuite();
        suite.addTest(new BookTest("testEquals"));
        suite.addTest(new BookTest("testGetPrice"));
        return suite;
    }

}
```