BEST: Journal of Humanities, Arts, Medicine and Sciences (BEST: JHAMS)
Vol. 1, Issue 2, Dec 2015, 69-76

UNIT TESTING *INVEST ON IT & SEE THE SUCCESS*

SONAL ROHILLA

Research & Development, Syscom Corporation ltd., Morpho, Safran

ABSTRACT

The paper is primarily focused on justifying Unit testing as an integral part of software development process. Few approaches of unit testing are touched herein, their pros and cons are discussed, which is better in which type of project. In the last section few options are suggested to ensure that unit testing is followed within a project team.

KEYWORDS: SCDL (Software Development Life Cycle), TDD (Test Driven Development)

INTRODUCTION

Stealing the words of Boris Beizer; quite sure people connected to software profession known who he is..:

"Software never was perfect and won't get perfect. But is that a license to create garbage? The missing ingredient is our reluctance to quantify quality."

Dear developers, honestly this is not to provoke you but to remind us of the responsibility of quality of product created by us.

Let's begin the topic

Software is a set of programmic logics created by an intelligent mind with the help of an intelligent language again created by super intelligent minds. Keeping in mind the complexity and human creation of software it can naturally be deduced not to be defect free.

Often in software industry we come across statements like if developer is testing the code then what is a tester suppose to do; though statements are wrapped in professional envelopes and not as plain as mentioned here.

Fully agreeing with the statement; a developer is suppose to write code and tester is suppose to test BUT this means that a developer is suppose to write a testable code and tester is suppose to ensure that this software works in maximum permutations of environment/input data/load & stresses.

Both the roles are mutually dependent; if developer gives the potentially workable/testable solution then of course the tester can focus in checking further set of combinations thereby ensuring the right proportions of quality.

The testing done by developer to check whether the piece of code is behaving the way it is intended to, is referred as Unit testing

The scope of this paper will be confined to unit testing approaches and bringing the same in project life cycle.

Test Driven Development (TDD)

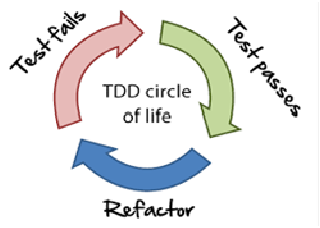

Figure 2

Unit testing can be started as soon as we start to write the code or even before that. Testing methodology defines a term 'Test driven development (TDD)' in which a test is prepared before even starting to write the code. Once test is done the functional code is written.

The motto of test driven development is "Red, Green, and Refactor".

- Red: Create a test and make it fail
- Green: Make the test pass
- Refactor: Change the code and improve the design while ensuring that all tests still passes

This cycle is repeated recursively for each new unit of code.

TDD is an approach which often proves to be beneficial for long term projects. Underlining benefits can be:

- It ensures that code actually works
- Refactoring allows improving the design of code successively. Hence the code proves to be better even in cases of short design phase.
- Early bug detection have very low cost of fixing
- Enables developers to change the code with confidence. Even if there is an error the test will inform them of issue.

In case of short term projects a project manager may think this as not a good option to return the considerable return of investment. Some of the drawbacks can be:

- Higher initial cost & duration in development phase
- Additional cost of test maintenance especially in case of change request or new short feature addition.
- Higher cost waste in cases where the project is withdrawn in development phase only.

STATIC ANALYSIS

Another option used for Unit testing is code review. Code review can be done by any peer developer especially a senior member of project team. For code reviews; in addition to manual review a good option can be to additionally use some static analysis plugins.

For eg: In case of java code, plugins like UCD, PMD can be used. These plugins can be easily integrated with

commonly used code development IDEs like eclipse.

List of plugins as per code development environment can be found at: wiki for List of static code analysis plugins

Advantages:

- Fairly well way to refactor the code to follow coding standards, improve code design and thus make it more maintainable.
- Cost effective solution, can be used in addition to other unit testing techniques.
- Disadvantages:
- No functional check of code
- Using same set of plugin over a period of time somehow restricts the focus of developer one defined set of rules.

Unit Testing using Automation Framework

When unit testing comes into picture; there are a wide range of test automation frameworks like JUNIT, Test NG, JWalk etc.

For ease, basics of Junit approach will be briefed here:

Junit is an open source framework for automating unit tests for Java. It comes as a plugin in eclipse.

In Junit, a unit test case includes a function which is to be tested, test data to test that function and steps to initialize that test case and exit that test case. It also includes method so that expected result can be matched with the actual result.

FEW BASIC FEATURES IN JUNIT FRAMEWORK ARE AS:

- In JUNIT, “TestCase” is an abstract base class, any test class will be a sub class.
- To write test cases, assert.class is used. assert.class is a library of classes which helps to match expected result with the actual result. Any discrepancy is reported as Failure.
- The execution of the test case is done through “TestRunner”.
- Each test case have three sub parts:
    - Setup() method: Preconditions required to execute the test case.
    - Test case scenario.
    - Tear Down() method: Test environment to be cleaned up for the execution of next test case. This is to make each case independent of the other.
- In the end; test results are logged using a color scheme. Green color for pass test case and “Red” color for failed test cases.

HOW TO START JUNIT

Steps to create Junit workproject:

- Open Eclipse IDE.
- Go To File – New - Java Project.
- Provide the name as “JUnit_Module”.
- Click on ‘Next’
- Go to ‘Libraries’ – ‘Add Library’ – Select Junit
- Browse for the JUnit.jar file. The JUnit.jar file would get added into the Project
- Select JUnit version i.e. JUnit(3 or 4) and then click on finish.

  Eclipse & Junit JAR is easily available over the net.

WRITING TEST SCRIPTS WITH JUNIT

Let’s learn how to create test scripts using JUNIT. Take a short piece of code:

```
public class employeeClass
	{
	String employeName;
	String employeDept;
	public boolean arequal(employeeClass E1) {
	if (E1. employeName == this. employeName && E1. employeDept == this.employeDept)
	{
		return true;}
	else
	{
		return false;}
}
```

- Go to Project (JUnit_Module) - src - Right click - Provide the name as “testPackage” and Finish.
- On “testPackage” - Right click and Click on Class option. Provide the name as “employeClass”
- Right click on testPackage – New - JUnit Test Case. Provide the name as “employeClassTest.java”

Following template gets created in eclipse:

```
package testPackage;
import junit.framework.TestCase;

public class employeClassTest extends TestCase
{
}
```

setup() and tearDown() methods can also be included during the creation of testCase.

```
package testPackage;
import junit.framework.TestCase;

public class employeeClassTest extends TestCase
{
	public employeeClassTest (String name){
	super(name);
	}

	protected void setUp() throws Exception {
	super.setUp();
	}

	protected void tearDown() throws Exception {
	super.tearDown();
	}
}
```

Now, to verify the name and dept present in employeeClass.java, junit.framework.Assert library will be used.

```
import junit.framework.Assert;
import junit.framework.TestCase;
public class employeeClassTest extends TestCase
{
	public employeeClassTest(String name){
	super(name);
	}
	protected void setUp() throws Exception{
	super.setUp();
	}
	protected void tearDown() throws Exception{
	super.tearDown();
	}
	public void testEqual(){
	employeeClass A = new employeeClass();
	A.Name="Jai";
	A.dept="Sales";

	employeeClass B = new employeeClass();
	B.Name="Karan";
	B.dept="R&D";
	Assert.assertEquals(false, A.arequal(B));
}
```

On execution, Assert.assertEquals (expected parameter, actual parameter) will match expected result with the actual result.

Executing test in junit

- Right Click on employeeClassTest.java and Select RunAs - JUnit Test

- User would get popup to Save and Launch dialog box , Click on OK
- JavaTest Runner would open indicating status of Test, either “Passed” (shown as Green) and “Failed” (shown as Red). It also shows time of execution of testclass in seconds.
- In case, there is Failed status then it will indicate <expected result> and <actual result>.

Nearly many test automation framework has many benefits like:

- Allows reusability of code
- Maximum coverage
- Detects bug at early stages hence low cost of bugs
- Framework is made such that additional efforts are minimized
- The framework is made in already used language so no additional domain expertise required
- Many frameworks can be easily used with tools like Selenium, QTP etc.

However there can be few points on downside as well:

- Not a preferable approach for short term project, small team size etc
- Many frameworks offer a much larger set of test combinations which at times becomes difficult to analyse and maintain in case of short term projects or less complex applications
- The complexity of framework increases in many cases like for keyword driven approach complexity increases with the increase in keywords.

CONCLUSIONS

Testing methodology, automation framework, static analysis plugins etc., offer a wide range of options for unit testing, out of which only few are mentioned here. The approach may vary depending on project duration, project domain, team size, team skill set etc but the main idea remains the same “Unit testing is an integral part of SDLC”.

Some amount of unit testing should be performed or promoted in every organization, nearly for every project. There can be various ways of ensuring that unit testing is performed in project life cycle like:

- A project manager or the project coordinator should build a process where the unit test cases (be it in any form) are archived in project archival database.
- Unit testing should be considered in initial planning and effort estimation. Thus a developer will be allocated a dedicated time to prepare unit test case and to unit test their code
- There should be a defined process to ensure that unit test logs are archived in the project archival database.
- Coding or code review guildlines should explicitly mention the usage of the static analysis plugins. Most appropriate ones should be choosed and shared within project team. Plugins can also be customized as per organizations coding guildlines.

- Organize trainings to develop a mindset and educate team for importance of unit testing.
- In end it can be just said that

"Unit testing will not ensure the absence of bugs but will sure reduce their occurrence"

ACKNOWLEDGEMENTS

I would like to acknowledge my co-workers & organization for supporting and encouraging me throughout the course work.

REFERENCES

1. Generically various testing basics over Google