Framework for Enhancing the Performance of Unit Testing in Java Based Projects

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted By

Name – Amit Sharma

(Roll No. - 801332003)

Under the supervision of:

Dr. Shivani Goel

(Assistant Professor)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2015

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Framework for Enhancing the Performance of Unit Testing in Java Based Project*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Shivani Goel* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(Amit Sharma)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Signature:

(Dr. Shivani Goel)

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. S. Bhatia)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success.

First of all I wish to acknowledge the benevolence of an omnipotent God who gave me strength and courage to overcome all obstacles and showed me the silver lining in the dark clouds. With the profound sense of gratitude and heartiest regard, I express my sincere feelings of indebtedness to my guide **Dr. Shivani Goel**, Assistant Professor, Computer Science and Engineering Department, Thapar University for her positive attitude, excellent guidance, constant encouragement, keen interest, invaluable cooperation, a generous attitude and above all his blessings. She has been a source of inspiration for me.

I am grateful to **Dr. Deepak Garg**, Head of Department and **Dr. Ashutosh Mishra**, P.G. Coordinator, Computer Science and Engineering Department, Thapar University for the motivation and inspiration for the completion of this thesis.

I will be failing in my duty if I do not express my gratitude to **Dr. S. S. Bhatia**, Senior Professor and Dean of Academics Affairs at the University, for making provisions of infrastructure such as library facilities, computer labs equipped with internet facility, immensely useful for the learners equip themselves with the latest in the field.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends who with their thought provoking views, veracity and whole hearted cooperation helped me in doing this thesis.

Amit Sharma

(801332003)

Abstract

This age is information technology age. We are living in a world where everything is handled by computers, smartphones and other software based systems. The dependency on software systems has increased to a great extent. The complexity of softwares has become very high these days since the software is supposed to serve for multiple functionalities. Earlier the softwares were not that complex and it was easy to develop the software so the software developers did not have much overhead regarding the product to be developed. As the complexity of the softwares increases the logic which is required to develop the software also becomes more complicated which ultimately results in a very complex code required for the software. The motive of this introduction is to explain that the coding part of the softwares has become very complex now. Developing and maintaining such complex code is not an easy task. In order to make sure the written code is working correctly and to make it reliable and sustainable, one of the software testing technique which is called as unit testing is used. As the complexity of the code has grown quite high, resulting in a very large code base, so the number of unit test cases which are needed to be implemented has become very high. In order to make sure whole code is working fine, all the unit test cases should be executed successfully before moving the code to the next phase of development cycle. Now the main overhead in this case is executing all the unit test cases one by one manually. Since there is a very high number of test cases, so if all the test cases are executed one by one manually then it will not only require lots of human effort but also it needs very high amount of time, which is not feasible.

This thesis presents a framework to make this process automated for java based projects. Using the suggested algorithm all the execution process of unit test cases become automated. After implementing the framework, the developer only needs to write code and the required unit test cases but there is no need to execute all the test cases manually. The complete execution process is taken care by the framework. This will not only save human effort but also reduces the execution time of the unit test cases which enhances the performance of development process.

Table of Contents

List of Figures

Chapter 1 Introduction

Before beginning the thesis let us discuss software engineering, since this thesis revolves around software only from the very beginning to the end. The definition of software engineering given by IEEE goes as, "*Software engineering is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software "* [1]. The software engineering takes care of all the processes require to develop a software system from scratch. The software engineering provides answers to all the questions related to development of the software:

- The engineering starts from very basic issue that what the problem is and how the problem will be solved?
- What steps will be takes to solve the problem [2]?
- How the stated problem will be converted to real world model and after that how this real world model will be realized as the actual product [3]?
- The software engineering also suggests very well defined process to verify whether the realized product is really the product for which this whole process, for which the whole work was done i.e. if the product is serving for the correct functionality for which it was developed or not [4].
- Suppose if it is not behaving properly then what steps are required to make it work correctly?
- Once the desired product is available how it will be managed to make it long lasting and how it will be maintained [3]?

Before starting the development of a software system all the steps of software engineering are defined and they are followed till the final product is realized. In simple terms software engineering is a series of sequential dependent activities through which a software is evolved, developed, tested and maintained throughout its lifetime [5]. This is also known as Software Development Life Cycle also abbreviated as SDLC. All the activities are performed in a sequence and the "Dependent" word is used because the output of one activity is used as an input for the next activity [6]. In order for a software to be developed properly and for the final product to be very efficient a software has to

go through all the steps of SDLC. The basic steps of SDLC consist of problem definition or problem analysis, design, development, testing or verification and after that maintenance of the product [5]. The maintenance step comes after the product has been released for the customer. Every steps itself consists of number of tasks which is discussed in brief later.

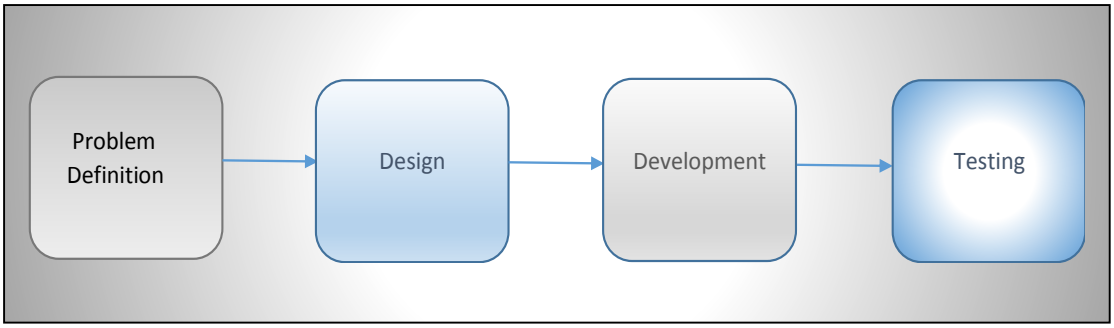


Figure 1: Steps of SDLC [5]

Figure 1 shows the four basic steps of SDLC. These steps are followed by all the organizations which are involved in software development work. Every organizations has it's certain standards but the baseline follows these steps only.

1.1 Problem Definition

This is the phase where the process starts. This phase mainly focuses on what is going to be done [4, 7]? This phase is also called as analysis phase. This phase basically includes all the analysis work which is required for the software development. This phase begins with requirement analysis task. In requirement analysis the customer is asked about his requirements from the product. This analysis is performed by analyst from organization and the requirements are collected by asking various question to the customer [4]. This phase plays an important role in software development since if the software engineer does not even know clearly what he needs to do then the final product will definitely be not good. All the requirements stated by the customer are collected and documented properly and which are given to the estimation team. The document in which the requirements are collected is known as Software Requirement Specification (SRS) document [5]. This SRS is a well formatted document in which the requirements are categorized into different categories depending on their nature. The estimation team performs a series of estimations based on the SRS which includes the budget estimation, total duration estimation, size estimation, effort estimation, resource estimation and the

seen and unseen risks involved in the project. So in problem definition phase the main focus is on what is going to be done? What resources will it take to get that done? What functionalities the final product should have? In brief all WHATS are resolved in problem definition phase [3].

1.2 Design

This phase focuses on How! Once the problem is clear after the requirement analysis phase the requirements are analyzed and then are converted into a real world software model. This phase does not talks theoretically but all the things are worked out from practical perspective [4]. The design phase mainly focuses on four aspect of the software development.

- The first one is data structure used for the software [5]. As per the requirements which data structure will be best suitable for the software which will take the least space complexity and will help the software to work in very less time resulting in very less time complexity. Data structure is decided by looking at the estimated size of the software. If inappropriate data structure is chosen then that may cause the whole system to behave incorrectly and which results in collateral damage.
- The second aspect which is taken care in design phase is software architecture. Software architecture defines how the software will look like from inside? What are the components and modules required to make the software work and how the components will communicate with each other and how the information is shared between various components? This is very complex information and these details are generally hidden from the customers. A number of design documents are proposed with different design style from experienced people and most efficient is chosen to be realized as final product. The design documents also suggests which component will serve for which functionality and they are designed in a way that all the components represent product as a whole and serves for the complete functionality [5].
- The third aspect is interface representation which defines external details of the software that how the software will look like to customers [4]. This phase defines the customer – software interaction details. This is done by analyzing the requirements given by the customer and by keeping in mind the technical

and other knowledge level of the customer i.e. what society of the people is going to use the product. Number of details are defined like what will be the basic graphic user interface (GUI) of the software, how the user can provide inputs to the system, the user interface should be pretty interactive and informative.

- The final and most important aspect is that all the details collected from customer, all the details analyzed for the product, how they all will be implemented? The fourth aspect is procedural detail or algorithm implementation [5]. This phase defines how the problem will be solved? What pathway will be followed to make the things working? Algorithms are implemented in this phase which are later converted into logical codes. The algorithms which are implemented are defined by keeping the data structures which were decided earlier and also the time and space complexity in the mind. The complexity of the algorithm depends upon how complex the functionality of the product is? If the model designed is very complex then definitely the logic required to realize that product will be very complex. This phase finally dictates how the product will be realized? The output of this phase is documented in Software Design Document which is used as reference document in the remaining life cycle of the software.

1.3 Development

This is coding phase. In this phase all the logics that were designed in earlier phase are implemented using some computer language. This is taken care by the developers. They not only implement internal logic using coding but also the interface is developed in this phase [8]. In which language the code will be implemented is already decided in earlier phase while implementing the algorithm. The language is selected based on the functionality required for the product and the data structure used to implement the software. The output of this phase is working copy of the software which is not the final but yes, it resembles to the final product up to a great extent.

1.4 Testing

This phase serves for checking for errors and problems with the implemented product in the development phase. Testing is basically ensuring that whether the implemented product behaves as it is supposed to behave or not? Testing ensures the product meets

it's functional, business and quality requirements [9]. Testing is performed by using the product from the perspective of customer who does not have in depth technical knowledge and may provide some unexpected input to the system which may cause the system to behave incorrectly or to get failed in an abrupt manner. If the product is only developed and not tested then the final product might have some errors which were left unnoticed when the product was developed. Software testing can be done once the development has been completed and it can also be done in parallel with the development process [5]. As the software complexity has grown very high and the implemented code is very large in size so it is not feasible to perform the testing after the whole code has been developed so the testing is done for small modules of code in parallel with the development. Testing is broadly classified in two categories which are White box testing and black box testing, which are further categorized in multiple categories. Each testing technique require different skill sets and method to perform testing but the final aim of all the testing methods is to improve the quality of the product [9]. The various testing techniques have been shown in below given table which shows their various characteristics like what is the scope of the testing, who performs the respective testing, what is the opacity of the testing i.e. if it is white box testing or black box testing.

Testing Type	Specification	General Scope	Opacity	Who does it?
Unit	Low-level design: actual code	Classes	White box	Programmer
Integration	Low-level design: High level design	Multiple Classes	White box; black box	Programmer
Function	High level design	Whole Product	Black box	Independent Tester
System	Requirement Analysis	Whole Product in Environment	Black box	Independent Tester
Acceptance	Requirement Analysis	Whole Product in Environment	Black box	Customer
Beta	Ad hoc	Whole Product in Environment	Black box	Customer
Regression	Changed documentation; High level design	Any of the above	Black box; white box	Programmer of Independent Tester

Figure 2: Software Testing Techniques [9]

1.4.1 White Box Testing

This testing is related with coding part of the software. This testing ensures that the implemented code does not have any errors, defects or some vulnerabilities which may have been left unnoticed during development phase. White box testing is generally done by developers only, this is because in order to perform white box testing one should have complete implementation details of the software and also should have good coding skills of the language in which the respective code has been implemented. The white box testing as per the name shows it is only concerned with the internal logic of the software, it has nothing to do with the functionality of the product i.e. it takes care of what is there inside the box [9]. The white testing is subcategorized into two categories which are unit testing and integration testing.

1.4.1.1 Unit Testing

Unit testing ensures that the code has no errors by testing small units of the code one by one. The main purpose of unit testing is that the code should not behave incorrectly even in case the user provides unexpected inputs [10]. The developer writes number of unit test cases depending on the source code and then tests the code against these test cases. Unit testing being white box testing is performed by developer only. In most of the cases the developer writes a piece of code and tests it. Sometimes some other person also need to perform unit testing on the code which is written by some other developer. In cases like these first the person who will be unit testing the code need to completely understand the code which has been developed and then prepare unit test cases for the code. Unit test cases mainly targets methods or procedures to be tested since these are the smallest unit a program can have [9]. Each and every method, control structures and all the statements have to go through unit testing to make sure they will not behave incorrectly with unexpected input and will not cause the whole product to behave incorrectly. The most important benefit of the unit testing is that it is done on very micro level so if there is any error at this level then it is very easy to trace it down and resolve it. Suppose if the code is not unit tested and some error has been left unnoticed then in later phases when the whole product has been implemented then this small error can cause the whole system to break down. On this level it is not easy to trace the cause error and resolve it since may be making changes in one component can affect other components because of internal dependency. Unit testing is performed on all the units

one by one and thus the whole code is tested. The size or number of units depend on which methods or control structures are needed to be tested. However it is a good practice to have unit test cases for all the methods but unit test cases are not written for all the methods. Most of the times unit test cases are not written for setters and getters since in these methods there is no calculations are performed only they are used for the purpose of either setting some value or in order to retain some value. Unit test cases are written for the methods where some calculations are being performed or some operation is performed and its result is to be returned. Suppose we have a class Test.

```
public class Test{
    String name = "Testing Variable";
    public void getName( ){
            return name;
    }
    public int multiply(int a, int b) {
            return  a*b;
    }
}
```

In this class we have one variable and two methods. First method is a getter method which is used to retain the value of private variable name and the second method is multiply which performs multiplication of two variables. Here it is pretty clear that there is no calculation is being performed in *getName* method, it is just returning a variable but multiply method performs calculation and then return the value of multiplication of two variables [10]. So we only need to test the multiply method. There can be number of test cases written to test whether the multiply method works fine for different inputs or not i.e. It should always return correct value.

For example we can consider the case that it should return the value zero when one of the variables or both of them zero. Other case can be if one of the variable is negative then it should return negative integer. Also if all the variables are negative or positive it should return a positive integer. Thus unit testing makes the developer code with confidence. It makes the code less vulnerable, more sustainable and long lasting which will not cause defects in later stages and will never break down.

Now a days as the code has become more complex so it very necessary to unit test the code. Since if the code is not tested at earlier development phases then it will be very complex to test this complex code after the product has been developed. That is why a new process called Test Driven Development, abbreviated as TDD, has come into scenario and is being followed extensively. Test driven development focuses more on quality of the code rather than the quantity of the code i.e. the priority is given to the testing. In test driven development as per the name first the developer writes test cases and later write implements code [11]. In this way the developed piece of code will always be free from errors and has very less chances of breaking down. The beauty of TDD lies in the fact that the unit test cases are written to get failed only. Since initially there is no code implemented so the unit test case will definitely get failed. So after that the code is implemented to pass the written unit test cases. Once the unit test case passes the code goes through refactoring process which includes implementing basic rules of OOPS like loose coupling, encapsulation, less inner classes etc. [11]. After refactoring the tests are executed again to check if the refactored code will pass the unit test case or not. If the unit test case fails then again the code is implemented and the whole process is repeated again. TDD helps to improve the code quality and reduces defects in the code which makes code long lasting and sustainable.

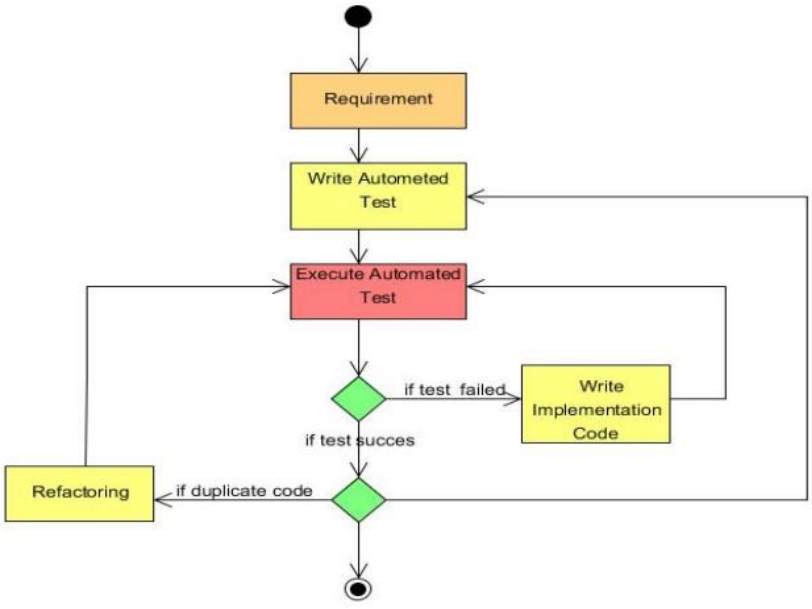


Figure 3: Test Driven Development Flow Chart [12]

1.4.1.2 Integration Testing

This is also performed by developer only. This is next part of unit testing. Once all the units have been tested then the integration testing begins. In integration testing the units are merged with one another and are tested for their correctness as a module [9]. This includes testing two or more classes or packages together that whether they are working fine as a module or not. This testing is very important in the fact that as a unit if a product is working fine then it is not necessary that even it will behave fine with other units also. Since when the units are merged there may be some internal code which is conflicting with each other and which results in interfaces to work incorrectly. So in integration testing integrated parts of software are tested for quality.

1.4.2 Black Box Testing

Black box testing is related to the functionalities provided by the software not about the code which is used to implement the logic. Black box testing is performed by individual testers or quality analyst persons who uses the software from the perspective of the customer. In order to perform black box testing one should have complete knowledge of functionalities which will be provided by the product [13]. Functional testing is further classified in following categories:

1.4.2.1 Functional Testing

The functional testing is performed by quality analyst people who first read the complete Functional Specification Document (FSD) which includes what are the requirements specified by the customer. After having full functional knowledge they checks whether the developed product adheres to required quality or not and if it is providing all the functionalities or not [13]. In manual testing first the tester writes test cases based on analysis of FSD and after that tests the product based on the test cases. If the product passes the test cases then it means it is providing the required functionality and if it fails the test case then the tester raises a bug or a defect and the product is sent back to development team for review or enhancement or some error.

1.4.2.2 Automation Testing

This is automated tool driven testing. A lots of tools are available today which are used to write test scripts based on the test cases written in manual testing phase and then execute these test scripts automatically [14]. Automation testing is the latest testing

which has made the procedure of testing automatic. In manual testing the tester has to execute all the test cases one by one but in automation testing the scripts are executed automatically. If the scripts get passed which means the test has been passed. A number of tools are available for automation testing like Quick Test Professionals (QTP), Synergy, Selenium, Open Script etc. Out of these some tools are open source while some are license based. Most of these tools are record playback based tools. In which first the tester records all the steps which are mentioned in the functional test case and then playback them for testing purpose with different inputs. The tool generates automated scripts based on the steps performed by the tester while recording. Now once the recording has done the script becomes permanent and it can execute any time you want and even it is possible to modify the script based on some other test case. Thus automation testing tools make the testing process automatic resulting in less time and more productivity.

1.4.2.3 Regression Testing

A product is not one time thing i.e. develop once and go home thing. It keeps going under changes and enhancements even after it has been completed once. Once the product development has been finished and it has been released in the market even after that enhancements are keep going for the upcoming versions. Regression testing is performed to ensure that if some new changes have been added to the product then they are working fine with already existing modules and functionalities [9]. Which means the newly added changes should not cause the already existing one to behave incorrectly and to get failed to fulfill their functionalities.

1.4.2.4 System and Load testing

These testing are system level testing. System testing includes checking whether the complete system as a whole is working fine or not. This is performed when all the components have been integrated with one another and now have been converted into a system. This tests the software for overall functionality that it should not fail to fulfill the requirements of the customer [9].

The load testing checks whether the system will be able to work fine even in extreme load conditions which are very high as compared to normal load. The load testing checks for the availability of the system in extreme conditions [9]. The example of load testing includes testing the websites for the number of users they can entertain at a time.

The essence of load testing is that the system should not break down if the load reaches beyond the defined load limit for the system.

1.4.2.5 Alpha and Beta Testing

These are the testing techniques which are performed by the customers only. Alpha and beta is the names given to the versions of the software which indicates that this version of the software is meant for the testing only. Alpha testing is performed by the customer at the developer's site [13]. In alpha testing a developer is always present with the customer but the beta testing is performed by the customer on his own site. In beta testing only customer checks the product for its functionality. No developer is present there with customer. Beta versions are launched in the market while the alpha versions are made available in developer's site only [13].

1.4.2.6 Acceptance testing

This testing is performed to check whether the product meets its business requirements or not. The requirements specified in Business Requirements Document (BRD) are checked in this testing [13]. This testing is performed by the business owners who check for physical tests and other tests that whether the developed product meets contract terms and other business requirements which were dictated during initial phases.

Thus these are the basic four steps involved in the development of all the softwares. There is also fifth step which is called support or maintenance step in which once the product has been released for the customer, support is provided to the customer in cases like installation of the product at customer's site, troubleshooting, maintenance of the product etc.

1.5. Unit testing frameworks for Java Based Projects

In order to unit test the softwares number of frameworks are available depending upon the nature of language in which the code has been implemented. Here we will discuss frameworks available to implement the unit tests for projects which are implemented in java language. Out of these frameworks some are used to implement the unit testing while there are some tools available which makes the unit testing process very easy and convenient one.

1.5.1 Unit Testing with JUnit

JUnit is one of the most famous framework which helps developer to implement and execute unit testing for java based projects. JUnit helps developer in writing unit test cases by providing number of assertions which are used to compare values with the values returned by the method to be called.

1.5.1.1 Writing unit test cases with JUnit

In order to get started with JUnit the first and foremost step is to know which naming conventions are used in JUnit while testing. However it is not mandatory to follow the naming convention but the standard naming convention which is followed is designed in a way that only by looking at the name of the method one can decide which method is being tested and for what test case it is being tested [15]? Suppose we have a method named *largstestOfTwo* which returns the largest number out of two integers. Then the first test method should always be called as *largestOfTwoTest* which should be used to test the basic functionality of the method. Now we can test for other conditions like if the first variable is larger than the name can be *largestOfTwoFirstLargestTest*. Similarly if we are testing for the case in which we will pass second variable as largest then the name can be *largestOfTwoSecondLargestTest*. There can be a test where both variables are equal then the name can be *largestOfTwoBothEqualTest*. Thus for a single method largestOfTwo we have three test cases in which the name of every method is conveying the required information that for what case the method is being tested?

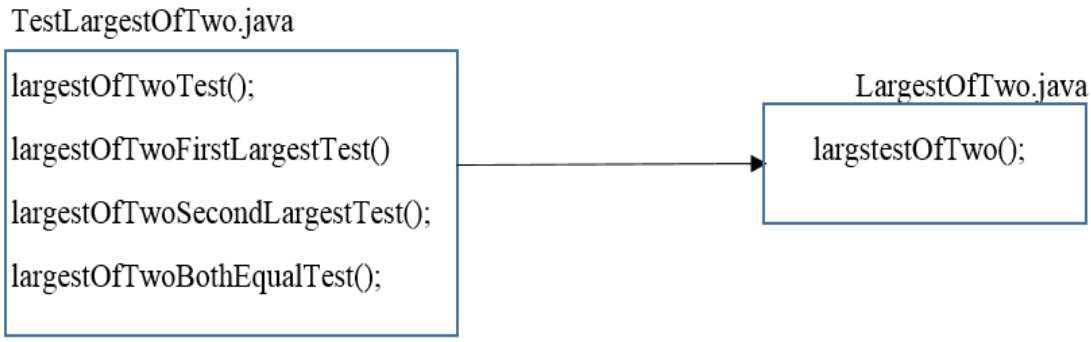


Figure 4: Testing and production code for sample java method

Fig. 4 shows both codes which are used in testing as well as production phase for the method *largestOfTwo*. Thus if the above mentioned naming convention is followed then the names of the methods become self-explanatory.

Other than naming convention the test code must be written to do a few things:

- Before executing test case all the pre-requisite conditions for testing should be fulfilled i.e. all the objects required for the testing should have been created in the test case, if some values are needed to be passed or some variables are there to be defined then that should be done while setting up the test case [15].
- The method which is being tested should be called in the test case based on the prototype of the method i.e. if the method needs some values to be passed then these values must be passed while calling the method. If the proper parameters are not passed to the method then the test case will fail for sure [10].
- Once all the setup conditions have been fulfilled then the test case can be executed. After execution the developer must verify that the function is working as it is supposed to be working.

We write test code and compile it in normal fashion, as any other bit of source code in the project. Just the difference is that the execution can happen in some different library but other than that there is no difference and no magic just regular code. When it comes to execute the code one thing is necessary to remember that here we are not running the production code directly; at least not the way a user would. Instead you run the test code, which in turn exercises the production code under very carefully controlled conditions.

1.5.1.2 Assertions used in JUnit

The basic purpose of the unit testing is to make sure that the method is behaving in the same way it is supposed to behave. This thing is made sure by calling the function with some input and verifying that it is returning the value which it should return. The JUnit provides assertions functions to verify the behave of functions that whether they are returning the values which they are supposed to return or not [15]. A number of asserts are provided by JUnit which are used to compare the value returned by the function to the expected value which should be returned by the function. Following is the discussion for the asserts provided by the JUnit:

1.5.1.2 1 assertEquals

This is the most used assert provided by the JUnit. This is used in the simplest unit test cases implemented in the JUnit. This assert simply matches the value returned by the method to the expected velue it should have returned [15]. The structure of the asserEquals is shown below.

assertEquals([String message], expected, actual)

assertEquals takes three arguments. The first argument is the string type of message which is used when the test case gets failed. The second and the third arguments are the expected and the actual value which is returned by the function when it is called with the random values. If the expected value and the actual value returned by the function are not same then the test case is said to be failed. The expected value is defined by the developer which should be equal to the value returned by the function. One more variation of asserEquals is shown below where the tolerance of the given function is defined.

assertEquals([String message], expected, actual, tolerance)

By including tolerance as a parameter what extent of difference the actual value returned by the function and the expected value can differ from each other. For example the given assertion below looks for the result which is actually equals to 3.33 but the main thing here is that it will only check the initial two places of the decimal.

assertEquals("Should be 3 1/3", 3.33, 10.0/3.0, 0.01);

Any kind of object may be tested for equality; the appropriate equals method will be used for the comparison. In particular, we can compare the components of strings using this method. Different method signature are also provided for all the native types (boolean, int, short etc.) and Object. Be aware that the equals method for native arrays, however, does not compare the contents of the arrays, just the array reference itself, which is probably not what we want.

1.5.1.2.2 assertNull

This assert method is used when the method which is being tested is expected to return Null value. If the method returns object other than null then the optional message is provided by the assertNull method.

assertNull([String message], java.lang.Object object)

assertNotNull([String message], java.lang.Object object)

The assertNull also takes three parameters out of which the first one is used to display the message if the expected and the actual values are not equal. This is similar to all the asserts.

1.5.1.2.3 assertSame

This method is used if the called method returns object and we need to match the returned object with the actual object it should return. This method will fail the test if the object which is returned by the called method and the object which it should return ideally are not same [15]. The syntax of assertSame method is shown below. *assertSame([String message], expected, actual, tolerance)*

Similar to assertSame there is a method assertNotSame which is used to check that the actual and expected objects are not same and it will fail the test if both the objects are same [10]. *assertNotSame([String message], expected, actual, tolerance)*

1.5.1.2.4 assertTrue

This is used to test whether a Boolean condition is false or true. If the condition is false then the test fails and if the condition is true then the test passes .It only takes two parameters. First one is similar to all asserts which is string message which is displayed if the test is failed and second parameter is the condition for which we are testing the method.

assertTrue([String message], boolean condition)

The counter part of the method assertTrue is assertfalse in which we want to check the condition to be false. If the condition is false then the test gets passed otherwise the test gets failed.

assertFalse([String message], boolean condition)

1.5.1.2.5 fail

This method is used to fail the test manually. This is useful in the cases when we want to make sure the control should not reach at certain part of the code such as for example if an exception is expected then control should not go beyond that [15]. On this case we

can use fail method which will fail the test case with some optional message provided by us.

fail ([String message])

1.5.1.3 JUnit framework

All the asserts provided by JUnit are cannot be directly called in a test class. In order to use these asserts we need to import JUnit framework in our program [16]. Once the JUnit is imported in the class all the asserts are automatically included in the program. An example is shown below which uses a simple test program written using JUnit. If we analyze the program step by step we find out that what each statement serves for.

In every JUnit test class the first line will always be to import the JUnit features in program. Next the *SampleTest* is the name of the class which is being tested. The SampleTest extends a class TestCase which is default JUnit class which has defined all the methods and constructors required to write JUnit test case. This class can be inherited in the test class or the features can be imported in the program. In further steps the test class consists of two tests which are testing two methods named addition and subtraction which performs addition and subtraction of two integers respectively and returns the result.

```
import junit.framework.*;
public class SampleTest extends TestCase {
    SampleTest sample = new SampleTest();
    public SampleTest(String method) {
        super(method);
    }
    public void testAddition() {
        assertEquals("addition unexpected values, 4, sample.addition(2,2));
    }
    public void testSubtraction() {
        assertEquals("subtraction unexpected values", 0, sample.subtraction(2,2));
    }
}
```

1.5.2 TestNG

TestNG is also unit testing framework used for implementing the unit test for java based projects. TestNG stands for testing Next Generation java projects [17]. TestNG also provides asserts and other methods to implement unit test cases. The main thing about

TestNG is that it is used to test complex system codes or business components based code which is complex to handle with JUnit. TestNG can handle cases where we need to write parameterized test cases in which we need to pass objects as a parameter [17]. JUnit does not has any provision for cases like this. Also TestNG provides more flexible naming conventions and other features which makes it more easy to use. TestNG can be added as a plugin to the IDE software which the developer is using. TestNG is added in the project properties of the system under test and then the unit test cases can be implemented for it. If the IDE software which is being used to write unit test cases code does not support TestNG plugin then it is not possible to execute unit test cases using that IDE software. The IDE should extensively support the features of TestNG.

1.5.3 Mockito

Mockito is not a framework for implementing unit test cases but it is used as an add-on for JUnit and other frameworks which are used to implement the unit test cases. This basically extends the features of the unit testing frameworks. Mockito is used to stub the objects and get the desired values from the method which are being tested. Mockito controls the behave of the objects that when a particular object calls a method what should be the result! The mockito is used when the objects have some dependencies. Mockito plugin can be added to the IDE and then just like JUnit the mockito framework can also be imported in the project and can be used to stub the objects.

1.5.4 Code Coverage Tools

Code coverage provides great help to developer in unit testing the code. Code coverage is defined as the percentage of code for which the unit test cases have been implemented. Code coverage is of two types:

- First one is Line coverage which is the percentage of statements in the program for which the unit test case have been implemented. This is calculated by assessing the unit test cases. It is calculated by assessing what number of lines is being tested by the unit test cases [18]. The line coverage basically includes the direct statements, declarations, methods etc.
- Second one is Branch Coverage which is different than the line coverage. Branch coverage is the percentage of control structures for which the unit test cases have been implemented [18]. Control structures include if-else conditions, branch statements etc. This is also calculated similar to the line coverage except

the fact that here control structures are included in calculation not simple linear statements.

Cobertura is an open source code coverage tool which is used to find out code coverage. Cobertura calculates code coverage and generates code coverage report. This report includes line coverage as well as the branch coverage.

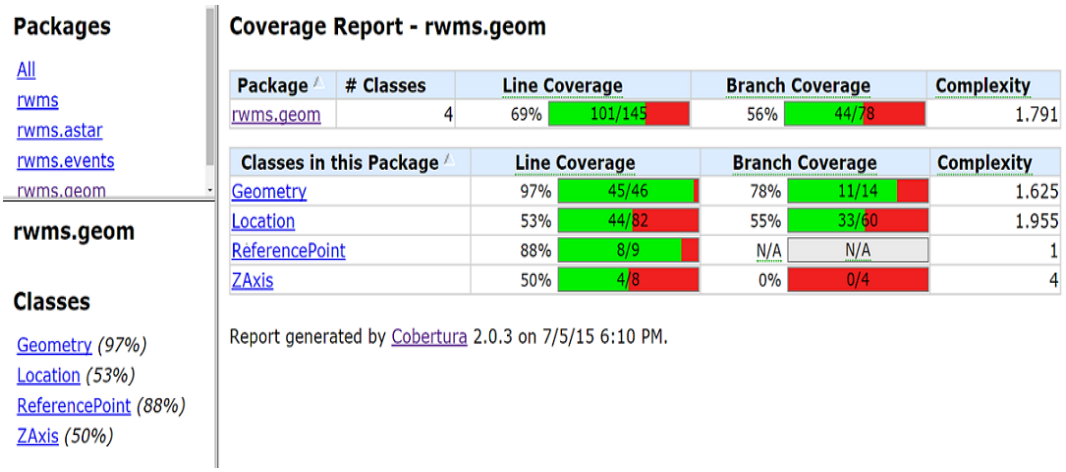


Figure 5: Code coverage report generated by Cobertura

The figure 5 shows coverage report generated by the cobertura. This consists of various information like which for which packages the coverage report is generated, what classes are available in that package and for what percentage of code in that class unit test have been implemented. It shows line coverage and branch coverage for all the classes in the package. In order to get this coverage report the cobertura plugin should be embedded in the ant build of the project. Once integrated it generates the coverage report when the build is executed.

1.6 Project Build

In simple terms a project build can be defined as another or a pre-release version of software. Build. This is just like building or constructing something for which the final result will be visible and concrete [19]. The build process mainly includes conversion of the source code to the final software vestiges which are similar to the final software which is used to run on a system. Builds play an important role in the software development process. A build basically takes all the components collectively and then compile all of them to make sure the final product is reliable. Figure 6 shows software development along with one more phase deployment phase.

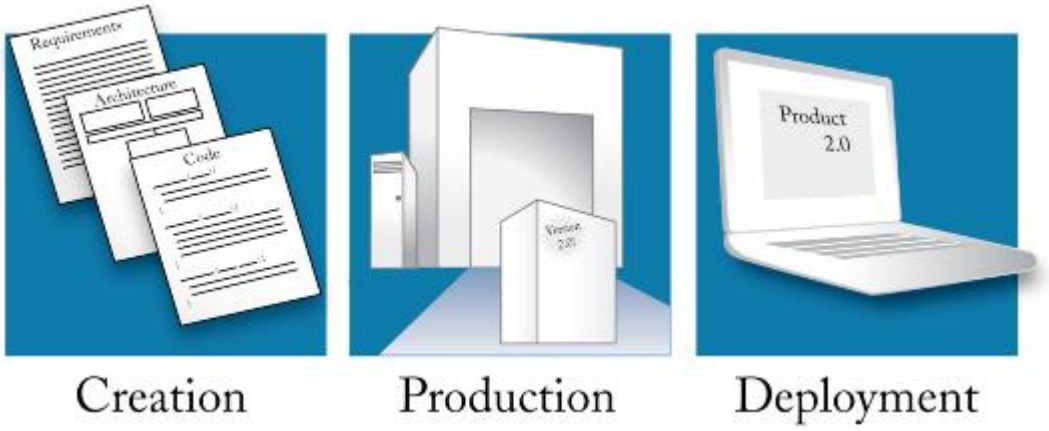


Figure 6: Software development steps along with deployment [19]

The deployment phase takes place when the source code is completely been developed and all the resources which is required to realize the final product are ready. Once all the resources are ready the build process of software can be started. This is called as deploying the build of the software. The benefit of build is that it collectively compiles all the components and source files of the software and make sure that everything is working fine. A build process performs all the steps of software development process, like development, integration and testing in single step [19].The final result of deployment should be a successful build which means all the components are working fine and no error exists in any components. If the build is not successful i.e. if it has been failed then there is definitely some error in the components of the software and it will not function correctly. Deployment of build includes mainly two steps. The first one is preparation or specification of the build and second step is execution of the build [19].

First step is specification of the build which includes specifying that in what steps the final system should be realized? A number of frameworks are available for creating the specification of the build. Some of them are Scons, Ant Script, Visual Studio etc [19]. A large number of options are available to define how a system should be constructed? The second step is execution of the build i.e. deployment of the product. For execution of the build some tools like Hudson, BuildForge and Cruise Control etc. are available [19]. Some of these tools are open source while some are commercial tools. Once the build execution is completed successfully, the software artifacts are generated. Talking about java based projects, the software artifacts include, .jpr files, jar files, static

analysis reports and various log files like system logs, application logs, server logs etc. which can be used for build analysis. Here in the implementation details build of Retail Warehouse Management System (RWMS) has been used to implement the required algorithm. Figure 7 shows Hudson tool which is used to kick off the build on the server. The most important thing about this build execution using Hudson is that all the components required in order to execute the build should be available on Hudson server. For this proper repositories are created for all the projects and all the components like developed source code, support files, build files which are developed in scripting languages and other files which are required during execution are needed to be stored on the server. So that when the build execution starts, all the components are collectively available on the server. In order to store the components on the server versioning tools like svn are used.

All the files which are required on the server to execute the build are checked-in on the server. The svn directly stays in sync with the server and the respective project hierarchy. Once the build execution is successful, all the artifacts generated in build execution, various logs and all the system files are available on the Hudson server only. All the files can be viewed and stored on the local system from the Hudson. The Hudson build execution tool is shown in figure 7. Once you are logged into the server you can see all the information related to the build files of the projects. All the build execution history on the Hudson in past time as well as the currently executing builds. You can see successful and failed builds along with the current status of the project which shows whether the builds of the project are getting failed or being executed successfully.

Hudson can be integrated in local environment or as a separate build execution server. Since the complexity of the Hudson is very high so generally it is not suggested to integrate it to local environment.

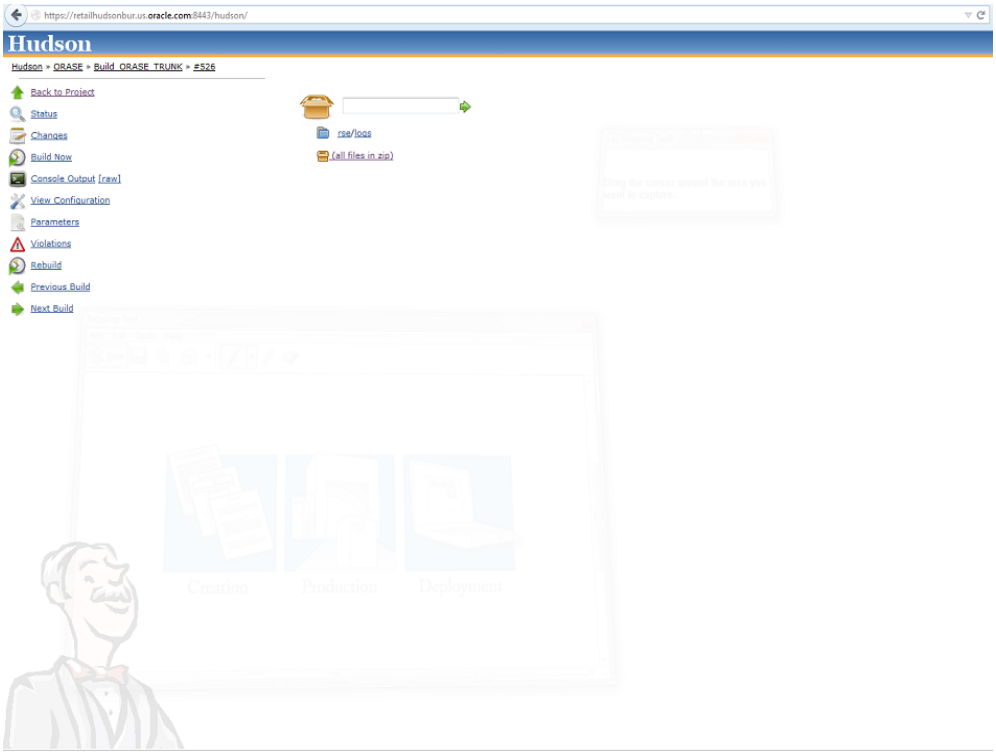


Figure 7: Hudson tool for build execution

1.7 ANT Script

ANT is product of Apache. Ant is an xml based scripting language which is used for specification of the builds for java based projects. Ant is an open source language. In order to deploy the project the build specification is first required. The ant is used to develop all the build files required for product deployment. In order to deploy the builds for java based project on the local environment, one needs to have Apache ANT installed on the local system. Once all the build files and other components of the project are ready the build can be deployed using the command prompt and ANT commands.

Unit testing has always been an important step in software development life cycle and there have always been research going on in order to design techniques which can be used to improve the performance of the unit testing. In 1988 D. Gelperin and W.C. Heztel classified the various phases of software testing evolution in the article published by them [20]. They classified the various phases on the base of time periods in which various techniques were used to test the software. From the starting of studies of software engineering to the 1956 there was no particular term called software testing but only the debugging of the code was performed in order to remove the errors and to make sure the software is working in a way it is supposed to. There was not a clear distinction line between the debugging of the software and the testing of the software. Actually at that time concept of software testing was not there. The next phase started from 1957 and went till 1978 in which the software testing concept was evolved. It was shown that only debugging the code will not make sure the software will work fine, there should be some better well defined process needs to be followed. Then first time the software testing was demonstrated which stated that the product should meet its requirements. Only debugging the product cannot guarantee a product with well quality. This phase created a clear line between the software testing and debugging. The software testing concept evolved very quickly in this phase. The next phase was classified starting from 1979 till the year 1982. The main aim of this phase was to detect as many errors as possible. This phase is also known as destruction oriented phase [20]. Reverse approach was followed in this age to test the software. It was considered that the software has errors and the only aim was to fins as many errors in the software doesn't matter what is causing the error. Whether it is the code of the product or it is because of some functionality which is not working properly. The next phase is defined from 1983 to 1987. This phase included the software testing as a phase in the software development life cycle in order to measure the quality of the product. In this phase various standards were defined which were to be meet by the product in order to be a good quality product. The final phase defined for testing is starting from year 1988 onwards and still going on. Here the primary goal of the testing was not to detect errors in the product or not to measure the quality of the product but it was to prove that the product meets its quality and business requirements. In this phase only the standard

definition of the software testing was provided by IEEE which is followed now also. According to IEEE standards the definition of software testing goes as, "The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects." [21].

2.1 Unit Testing

In 1971 Richard Lipton presented a paper demonstrating Mutation testing which was one of the practices of unit testing for testing the small part of the code at a time. After that in 1994 Kent Beck first time wrote a unit testing framework which was used for the softwares which were implemented in SmallTalk. The unit testing was new concepts in those days and it was still going under evolution. The first time unit testing framework for object oriented language, Java, was developed by Beck and Gamma in 1997. It was enhancement of the Keith Beck's earlier developed framework for SmallTalk projects [22]. After that many frameworks for unit testing were developed like Mockito, which was used for Object Stubbing process to get the desired behave out of objects.

2.2 Test Driven Development

In earlier 90s the softwares developed by the organizations were not that stable. Either they were delivered out of schedule or the product which was developed did not used to provide the functionality it was expected to provide. The softwares which were delivered to the customer either did not have long life. Some of the software will not even loads the basic feature or took long time in loading process. Thus the overall repo of software industry was not in pretty good condition [23]. There was definitely some problem with the process being followed or proper quality measures were not being taken. To find out what was the problem in 2001, some experts from software industry gathered to find out what was the reason of failures of projects? Why all the projects were either cancelled or were over-running the defined schedule? Why the delivered product did not meet customer's expectations? What was the defect in the process being followed? The output of this gathering was a new methodology called as Agile development process which defined a totally new development process with lot of modifications in existing process. Many new core practices were suggested. Test

Driven Development is also one of the practices which was emerged out of Agile development process [23]. Test Driven Development, also abbreviated as TDD, is the process which says "First test then develop". Kent Beck and Cynthia Andres demonstrated that using TDD in the development process number of benefits can be gained [23]. The benefits of TDD as mentioned by Kent Beck and Cynthia Andres are as following:

- In this practice first the test case is written and then the code which is implemented is developed with only one goal, to pass the test case. So the developed code has very limited scope. Thus TDD does not let the scope of the code to keep growing [23].
- TDD helps to identify if there are any issues in the design process. The process of writing the test cases for the code, which is highly cohesive and loosely coupled in nature, is very easy. The test cases can be written very easily for this code. Now if the task of writing test cases has become very tedious and complex then it is certain that there exists some issues in the design process of the code [11].
- It brings confidence in the various entities of the organization, like developers, stake holders, managers etc., about the quality of the code. Since the code which is demonstrated will always be written in order to pass a test case and it will be sent to next phase only if it passes the test case. So the code is always well written, highly reliable which is proved by existence of properly executed test cases [11].
- The most important benefit it provides is for the developers who love to do extreme programming i.e. using highly dynamic concepts in the code. Using TDD the developers can follow a proper step by step like first test, then write the code and then if required refactor it. All these steps provide a development environment which is not only productive but also it makes sure the code is highly sustainable [23].

2.3 Code Coverage Tools

Code coverage tools are also a very important part of software testing. Using code coverage tool software engineer can find out what part of code has been assessed by the test cases. This helps the developer to know still what part of the code can have errors

and needs to be assessed by the test cases. Now automated code coverage tools are available which makes the whole process and automatically calculated the code coverage of any project. Khalid Alemerien and Kennneth Magel studied the effectiveness of the testing coverage tools that how automated code coverage tools can increase the performance of development process. They studied number of tools available for code coverage and presented the analyzed data.

Tool Name	Type	Coverage Level	Instrumen -tation	Reporting	Integrated with JUnit	How to appear in eclipse
Ecobertur a	Open Source	Line, Branch	Byte Code	Direct in eclipse	Yes	Cover as
Eclemma	Open Source	Statement	Byte Code, on the fly	Direct in eclipse, Text, HTML, XML	Yes	Coverage as
Clover	Commer- cial	Statement, Branch, Method, element, Line, class, and contribut-ion	Source Code	Direct in eclipse, PDF, HTML, XML	Yes	Clover
Djunit	Open Source	Line, Branch	Byte Code	Direct in eclipse, HTML	Yes	Djunit Test
Code Pro Analytix	Freeware by google	Class, method, line, statement, branch	Byte Code	Direct in eclipse, Text, HTML, XML	Yes	CodePro

Figure 8: Comparison of code coverage tools [18]

The above table shows all the automated code coverage tools which were studied by Khalid and Kenneth. Automated code coverage along with automated unit testing makes the unit testing phase of software development very easy and less time consuming process. The code coverage tools are compared on basis of various parameters.

- The first parameter is licensing information of the tool that whether the tool is open source or commercial tool
- Coverage level means what types of coverage is provided by the tool. Line coverage is the number of lines assessed by the test code. Branch coverage is the percentage of control structures assessed by the test code. Statement coverage includes all the declarative statements in the program. Class and method coverage is the percentage of classes and methods for which the unit test cases have been implemented [10].

- Instrumentation is defined as the files on which the coverage tools perform assessment. Like some tools work on the compiled source files or byte code whereas some tools assess using the source code of the program [18].
- Next comparison is based on whether the tool can be integrated with the unit testing framework JUnit or not.

2.4 Automated unit testing tools

Unit testing automation was achieved by various tools and frameworks. In recent years the software development process are going towards an automated trend. All the processes followed in software are being converted to automated processes. All the phases of software development process are completed using some automated tool or framework. Similarly the unit testing is also going automated way. A number of techniques have been designed which makes the process of unit testing easy and more convenient.

Laurie Williams, Gunnar kudrjavets and Nachiappan Nagappan presented the framework for automation of unit testing process which is being followed at Microsoft Corporation [24]. A framework called NUnit was used to automate the process of unit testing. The experiment was performed for one year to check whether the automated process gives benefit to the existing process or it is not worth of adaptation. The unit testing process was made automated and the Test Driven Development was followed as a development process. The developers were asked to write the unit test cases using the unit testing frameworks once they have finished coding. The team was working on a version 2 of some product for which the development of version 1 was already has been completed. When the defects of the version 2 were compared with the number of defects in version 1 it was found that the defect rate has been decreased by 20.9%. However if the TDD is followed as a development process then definitely the overall time increases since first the test cases are written and then the code is implemented to pass the test. The development time was increased by 30% but the code which is developed using TDD is of very good quality code, well-written and more sustainable [24]. When the automated unit testing technique was employed, and the automated test cases were written repeatedly then there was great decrement in the number of defects. The figure 9 shows the analysis of various parameters for before and after employment of the

TDD and automated unit testing was employed. Once the automated unit tests were created, the number of defects with high severity were reduced up to a very good number in version 2 as compared to version 1 where the unit testing process was not automated. High severity and low severity is defined by the impact the defect cause on the functionality of the product.

	Windows	MSN	Visual Studio
Test LOC [3] / Source LOC	0.66	0.89	0.35
% block coverage	79%	88%	62%
Development time (person months)	24	46	20
Team size	2	12	5
Relative to pre-TDD:			
Pre-TDD Defects/LOC	X	Y	Z
Decrease in Defects/LOC	.38X	.24Y	.09Z
Increase in development time	25-35%	15%	25-30%

Figure 9: Case studies for TDD in Microsoft [24]

If the impact is very high then it is severity 1 defect and the severity level goes down up to level 4 depending on the impact of the defect. The table shown in fig. shows the difference in the number of defects in the version 2 and version 1 based on the severity of the defects. This shows there was great percentage of reduction of defects in version 2.

Defect Severity	Version 1 (%)	Version 2 (%)
Severity 1	15.5%	16.8%
Severity 2	49.8%	40.1%
Severity 3	28.7%	18.8%
Severity 4	6.0%	3.4%

Figure 10: Distribution of defects based on severity [24]

The automated unit test process not only increased the code quality but also it provided the developers a very good automated development environment. Microsoft also presented the results of a survey performed to check what impact the change in process brought to the perception of the developers and the result was highly positive. The developers found this very helpful for extreme programming

as well as they felt it helpful for development of a more reliable code which is free from errors. Figure 11 shows the survey conducted on the developer's perception after using automated unit testing process and Test Driven Development.

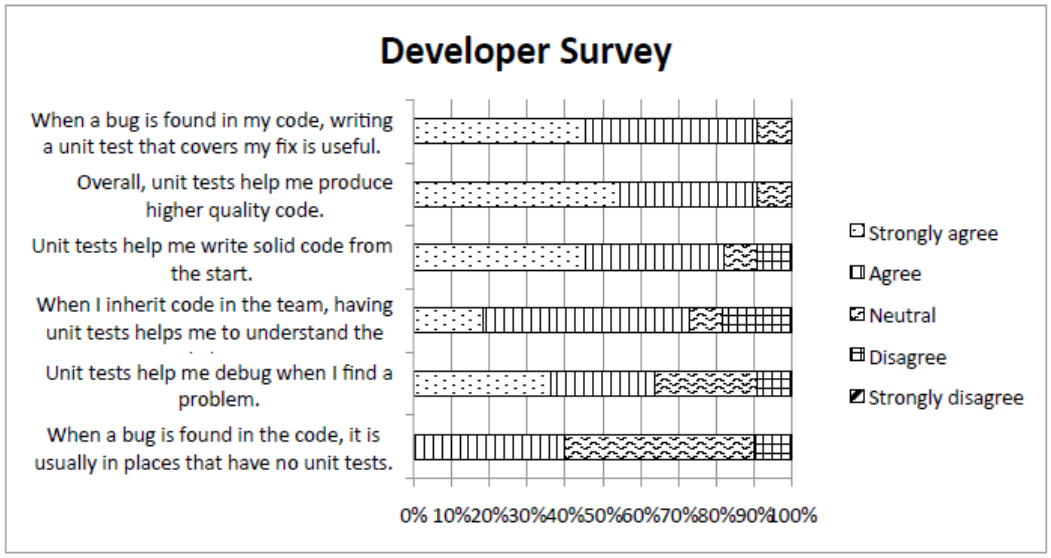


Figure 11: Survey of developer's perception [24]

Andrew Patterson, Michael Kolling and John Rosenberg presented a new method for unit testing automation. They performed amalgamation of two standard unit testing frameworks JUnit and BlueJ [25]. It was felt that for beginners JUnit is not that adequate because it has well defined processes which one needs to follow. So these three combined the very easy to use and flexible features of BlueJ with the JUnit. This method was mainly for the introductory people who either did not have knowledge of unit testing or had very little knowledge. The fact about JUnit that they provide very wide scope to write the unit test cases but they do not provide any support for writing the test cases. The issue with BlueJ is that it cannot be used for regression testing since there is no recording facilty provided in this tool. So with the integration of these two both functionality will be provided by one tool. The main task or changes which were needed to integrate both the tools were to make the BlueJ familiar with the capabilities of the JUnit. Now once the JUnit functionalities have been integrated into the BlueJ, the user can write test cases using JUnit classes and assertions but the interface to write these test cases is provided by the BlueJ which is very simple and interactive. The BlueJ provides very good interface for all the steps required for unit testing e.g. from beginning where a user creates test method and after that in asserting the results and finally executing test cases. If the user has little knowledge about JUnit test classes then

the BlueJ interface walks him through all the required steps. In JUnit all the test classes and methods are required to be written by the developer whereas when the JUnit and BlueJ are integrated then the user can write the JUnit test cases but in BlueJ's easy way. The figure 12 shows the interface provided by the BlueJ to write JUnit test cases for all the steps.

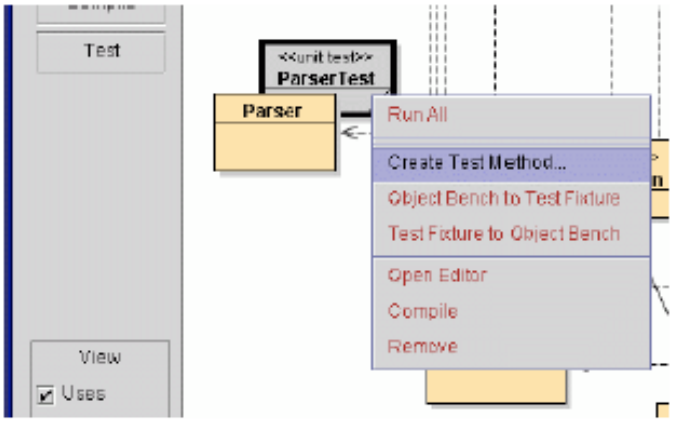


Figure 12: Creating test method in BlueJ interface [25]

Figure 12 presents creating a new test method with name *ParserTest* in JUnit environment but this interface is provided by BlueJ. Assertion of the results and execution of the test cases is shown in figures 13 and figure 14 respectively.

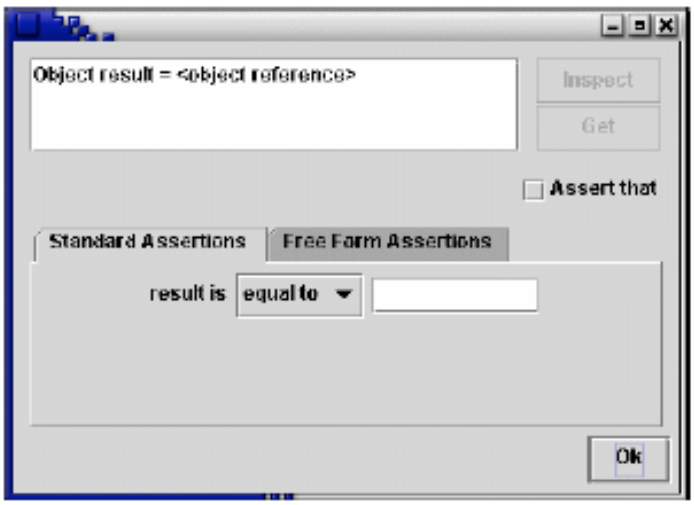


Figure 13: Asserting the results in BlueJ [25]

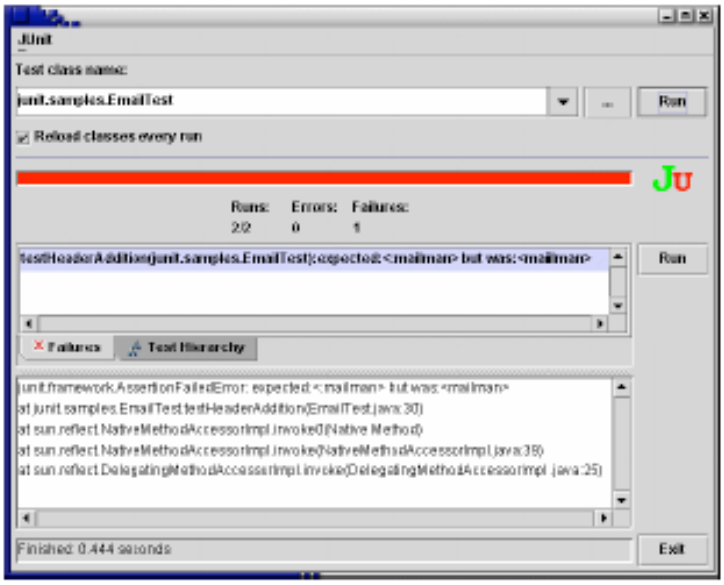


Figure 14: Executing the test cases in BlueJ [25]

Thus the integration of JUnit provides a very nice featured environment to the user with all the capabilities of the JUnit and convenient interface of the BlueJ.

The software is composed of multiple components. All the components are collectively integrated to make the product perform the required functionalities. The components used can be off the shelf components which have never been used and are designed for the current product, on the shelf components are those which have already been used in some existing product and are being reused, and the third category is third party product which are used by the organization but are not licensed by the organization i.e. they are either open source or are purchased from some other organization. The performance of the software depends on all these components. If some components require some modification then it should be done on unit or micro level and it should be tested properly. Ravinder kumar and Karambir Singh suggested framework to enhance the performance of component bases testing with JUnit in Net Beans environment [26]. A java based program was developed in the NetBeans platform and JUnit was used to implement the unit test cases for the program. The combination of JUnit and Net Beans was chosen because the Net Beans extensively supports JUnit. So there was no need to resolve integration issues of both frameworks like in the earlier case where JUnit was integrated with BlueJ. Unit test cases were implemented for all the components in Net Beans environment with the help of JUnit classes. The basic system flow used by them is shown in figure 15.

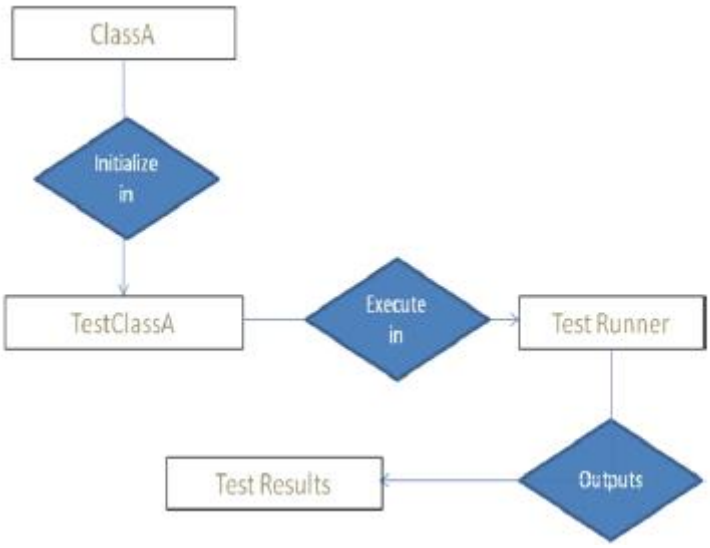


Figure 15: Testing process using JUnit [26]

Figure 15 shows the steps required in order to test a class using JUnit in java based IDE softwares. *ClassA* is initialized in *TestClassA* in order to be tested. Initialization means to declaration of variables, assigning memory to resources, declaring objects etc. Now this *TestClassA* is executed in the Test Runner class which is provided by the JUnit. Once the execution completes the results are collected and are verified for the required result. The basic purpose of this paper was to discuss the various features of unit testing framework JUnit version 4 in the Net Beans and show the effectiveness of JUnit in implementing and executing the unit test cases in java environment [26].

Unit testing is important for all the kinds of system since it resolves the error at very micro level. Similarly unit testing is also necessary for the legacy systems. The legacy systems are very large systems which already have been implemented and very less documentation is available for that system. Creating unit testing for the legacy systems are necessary since they can be used as a component for some new system and if there is some error present in the system then it will cause the new system to behave incorrectly [27]. Emad Shihab, Zhen Ming jiang, Bran Adams, Ahmed E. Hasan and Robert Bowerman presented method to implement unit test cases for the legacy systems [27]. The legacy systems are very large and not a lot of implementation details are available for them. So in order to implement unit test cases for legacy systems, they followed Divide and Conquer approach. They separated the methods of the legacy systems and then implemented unit test cases for all of them. Writing unit test cases for small modules is always an easy task. This approach was addressed as Test Driven

Maintenance by them. Since the legacy system has alresy been implemented so the process followed by them to extract the information of the methods for which the unit test will be implemented is shown in figure 16.

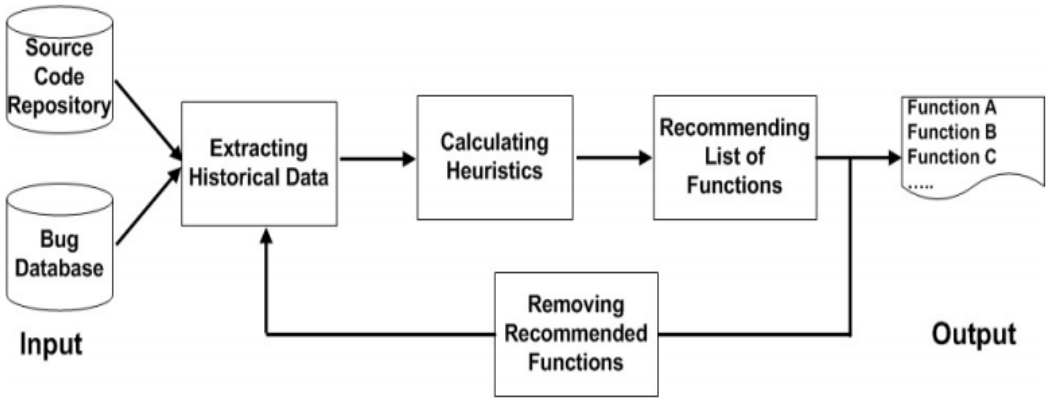


Figure 16: Flow diagram to extract the functions in legacy system [27]

Since writing unit test cases for a very large code base which has already been implemented is very nearly impossible. So there is a need to analyze the code base and write unit test cases for some selected functions. First of all the bug database for the legacy system is analyzed that which modules caused the most number of bugs and then those modules are extracted from the code base of the system. After the historical modules have been extracted from the code base, these modules are used to calculate various heuristics which are further used to prioritize the list of testing function. Based on the heuristics the list of the functions is generated which will be unit tested [27]. Once the list has been generated that function is removed from the pool of the function so that this function should not come under the list which will be generated in future using the historical data. So now all the methods are decided for which the unit tests will be implemented. Now unit test can be implemented for all the methods individually. This approach can be used for the legacy systems for which the unit tests have not been implemented while development.

Now the complexity of the softwares which are implemented in information technology industry these days has been increased very much. In order to develop complex softwares the coding practices employed are also very complex. If the code is very complex then it is necessary to perform some high level of testing since there may be lots of errors present in the code which might have left unseen. So unit test cases required for this code are very complex. Now as the functionality for which the code is being designed is very complex so the code base is also very large. There are almost thousands lacs of LoC. If the unit testing is implemented for this code base then the number of test cases are also very high. In order to implement the unit testing for java based projects one needs to have frameworks like JUnit or TestNG integrated in the IDE in which the code developed. Now what steps followed in implementing the test case are that first the developer writes unit test cases for the project, stores them is some project level hierarchy and then execute them. Here the steps to execute unit test cases in JDeveloper are shown using JDeveloper screenshots for RWMS hierarchy.

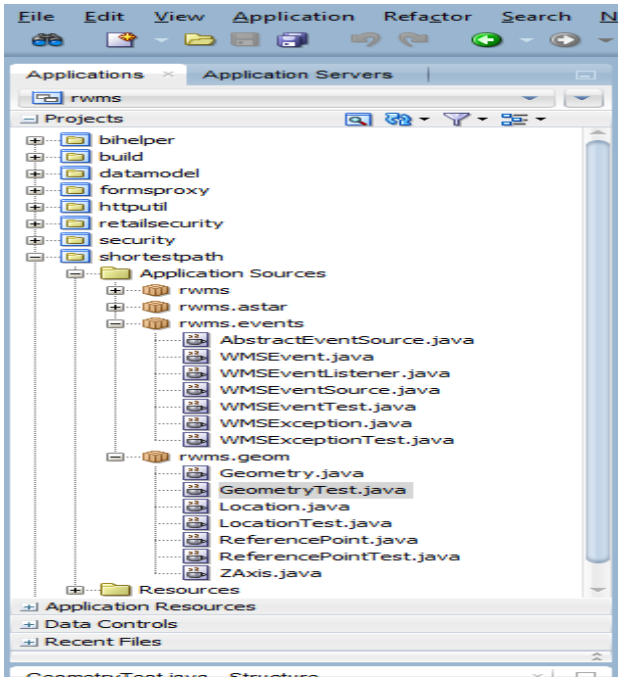


Figure 17: RWMS hierarchy

Figure 17 shows project hierarchy of Retail Warehouse Management System (RWMS) project. This shows all the packages and build folder in the RWMS. Now in order to

execute unit test cases one needs to add JUnit library in all the project libraries. Figure 18 shows addition of libraries JUnit and Mockito in RWMS project.

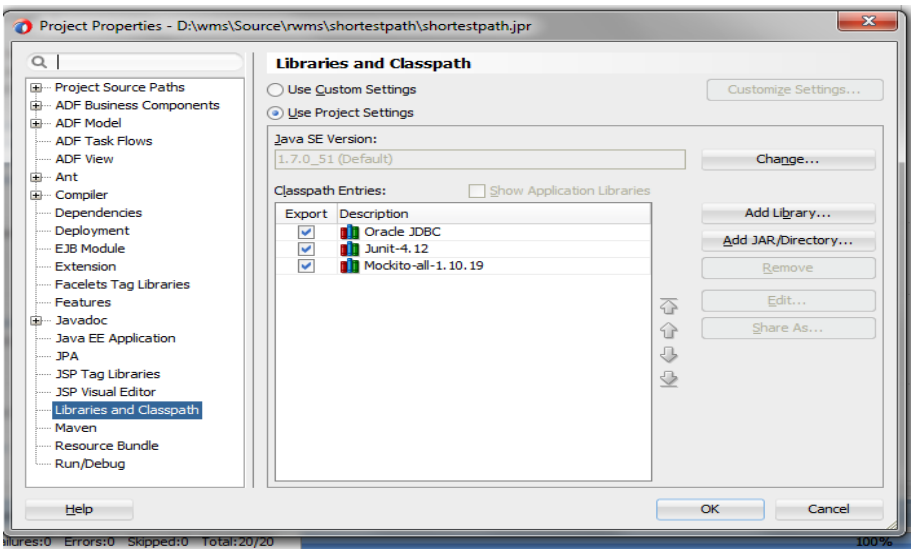


Figure 18: JUnit and Mockito integration in project libraries

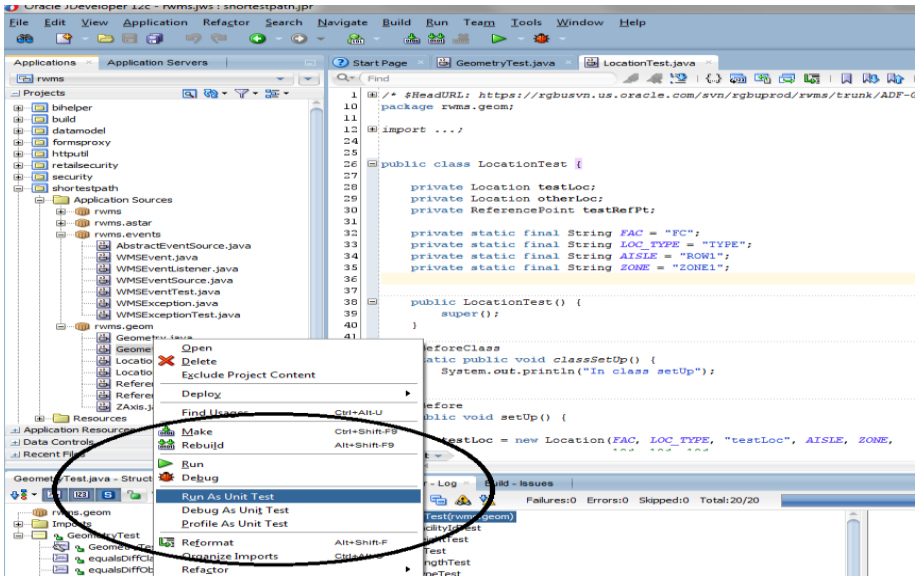


Figure 19: Running unit test cases in JDeveloper IDE

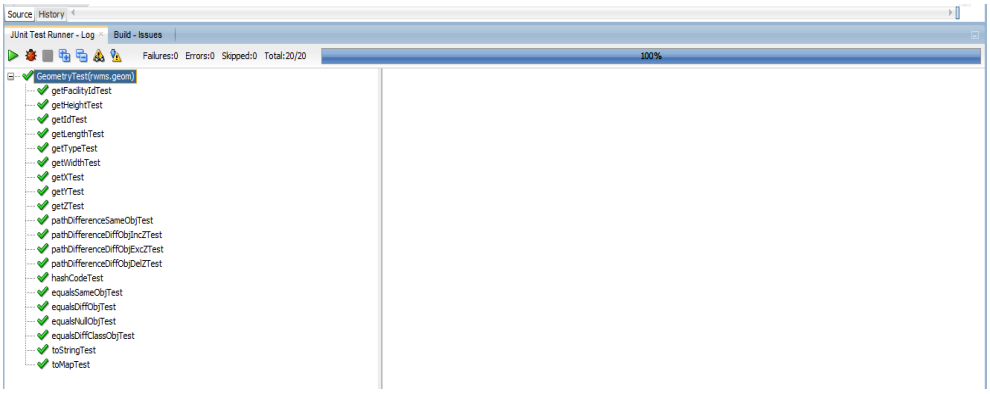


Figure 20: Test case execution results

So these are the basic steps followed to execute unit test cases for any projects using IDE. Since the codebase size increases so the number of packages increase and resulting in very high number of unit test cases. Since at most of the places the manual method to execute the unit test cases is followed. So if the number of unit test cases is very large then executing them manually executing the test cases will not only require high amount of time but lot of human effort is also required. However organizations cannot avoid this step since in order to make developer code with confidence unit testing is one of the most important step. Before the code goes to next phase all the unit test cases are executed one by one in order to ensure the integrity of the code. So this may become an overhead for the project since it requires both time and human effort. So there is a need of some automated system which can execute the unit test cases automatically and save this entire time as well as human efforts. So the basic problem for unit testing goes as:

- The number of unit test cases have grown very large as the code base has become complex. To test the code very high number of test cases are required.
- The developer not only needs to implement the unit test cases but also he needs to execute all the test cases manually one by one.
- Executing all the test cases with manual process require high amount of time which is not feasible for a software project.
- This manual process is not feasible for execution of all the unit test cases. There is a need of some automated process to handle the execution of unit test cases of the process.

Chapter 4 Implementation

In order to enhance the performance of unit testing for java based process there is a need to handle the execution of unit test cases in an automated way instead of executing all the unit test cases manually one by one. The optimized way to do this automatically is to execute all the unit test cases with the build process of java project. When the deployment of the project takes place all the unit test cases get executed automatically thus saving all the time and manual efforts required while executing them manually. To execute unit test cases with the build process of the project there is a need to integrate unit testing framework with the build specification. It is also required to add the location of all the unit test cases implemented in the specification of the build. The implementation for executing unit test cases automatically is shown by using Retail Warehouse Management System (RWMS) build. RWMS is also a purely ADF and java based project which has unit test cases implemented in JUnit.

4.1 Build process of RWMS

The build hierarchy for RWMS is shown in figure 21. There is one main build file which internally calls all the build files. Each and every build file serves for different purpose, which is out of scope of this discussion. All the build files are in .xml format which is by default format of the files which are developed using ANT script as ant is an xml based scripting language.

All the tasks which are to be performed by the build file are performed by creating targets in ant script. These targets perform various operations based on the script written for them. The functionality of target ranges from calling some other builds to executing java files and other components of the project. So in RWMS all the build files are being called in main build file in which the main operation of deployment is specified.

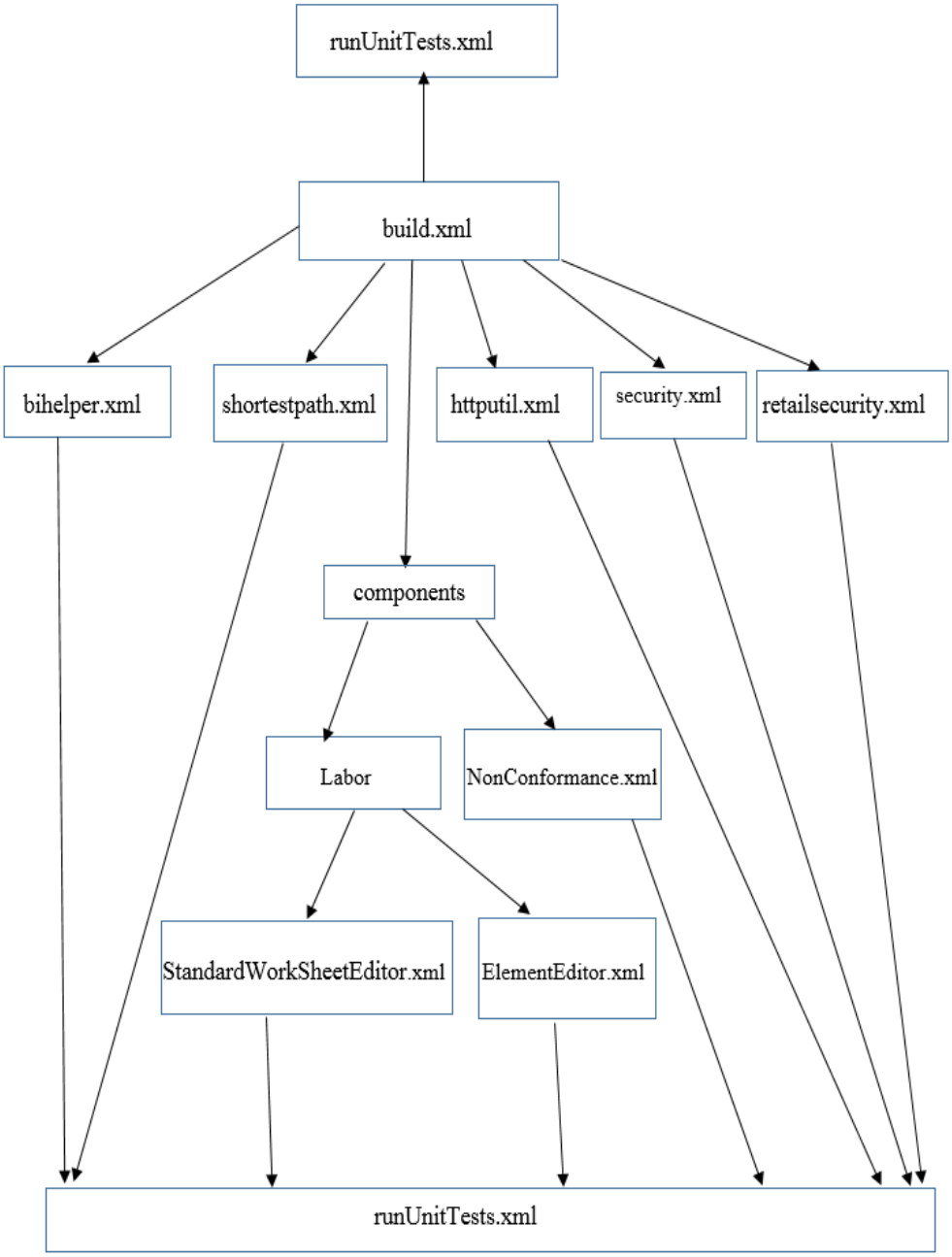


Figure 21: Build hierarchy for RWMS

In build.xml file target named *project.target.invoke* is used to call all the files. The name of target can be anything but the fact is that it is good to follow naming conventions which conveys some meaning for the given name. The code snippet for *projects.target.invoke* is shown below.

```
<target name="project.target.invoke">
    <echo message="Current task is (${target.name}) in *** ${project.name} ***"></echo>
    <ant dir="${projects.dir}/${project.name}" target="${target.name}" />
  </target>
```

This target calls all the builds like bihelper, httputil etc. In order to call those builds this takes parameter *project.name* parameter as an input. This task is performed by one more target which passes the name of the builds as a parameter while calling this target. This target is named as *projects.target.invoke* who performs the basic functionality of calling the target *project.target.invoke* by using antcall feature provided by ant. When one target calls another target using antcall the calling target also needs to pass the parameters to the target which is being called. The code snippet for *projects.target.invoke* is given below. It is passing *bihelper*, *httputil*, *security*, *shortestpath*, *retailsecurity*, *nonconformance* and two other components of labor which are *ElementEditor* and *StandadWorksheetEditor* as parameters in the call to *project.target.invoke*. The called target takes these parameters as inputs and then calls the respective build file from the main build file and then the control is shifted to the called build. Once the operations defined in the called build file are completed, the control is returned to the calling target that is *projects.target.invoke* in main build file. After that the rest operations in main build file are completed.

```
<target name="projects.target.invoke">
    <antcall target="project.target.invoke">
      <param name="project.name" value="shortestpath"/>
    </antcall>
    <antcall target="project.target.invoke">
      <param name="project.name" value="security"/>
    </antcall>
    <antcall target="project.target.invoke">
      <param name="project.name" value="bihelper"/>
    </antcall>
```

```
    <antcall target="project.target.invoke">
        <param name="project.name" value="httputil"/>
    </antcall>
    <antcall target="project.target.invoke">
        <param name="project.name" value="retailsecurity"/>
    </antcall>
    <antcall target="project.target.invoke">
        <param name="project.name" value="components/Labor/LaborElementEditor" />
    </antcall>
    <antcall target="project.target.invoke">
        <param name="project.name"
            value="components/Labor/LaborStandardWorksheetEditor" />
    </antcall>
    <antcall target="project.target.invoke">
        <param name="project.name" value="components/NonConformance" />
    </antcall>
</target>
```

This is how all the builds of RWMS gets executed. The control starts from main build.xml file. First all the operations of main build are performed and after that all the build files are called one by one. In one call the control is shifted to that build file and once all the operations are performed successfully, the control comes back to the main build file.

4.2 Integrating Unit Testing Framework in Build Process

This is the most important task in order to optimize the performance of unit testing of the project. First of all a unit testing framework which was used to implement the unit test cases for the projects is needs to be integrated into the build process of the project. After that it should be able to execute the unit test cases for all the builds of the project. As discussed earlier there are number of unit testing frameworks available for implementing the unit test cases like JUnit, TestNG etc. Depending on the framework used in the project to implement the test cases in the project, that framework needs to be integrated in the ANT build of the project. All the frameworks can be integrated in

the ANT build. Since in RWMS the unit test cases are implemented using JUnit, that's why JUnit framework has been integrated into the ANT build of the RWMS. JUnit is added into the build process by adding JUnit task or target in the builds. There may be many methods to integrate JUnit in the build process of RWMS. The best method which not only help to reduce the build overall execution time but also performs the required operation exactly it is supposed to be performed is chosen.

- The first possibility is to integrate the JUnit in main build.xml file which calls all the build files. In RWMS each build serves for a sub project while the main build serves for overall RWMS project. Each sub project has java files and for which unit tests are implemented. If the JUnit is integrated into main build then what the process will look like is that first the main build calls all the builds where the deployment of sub-projects take place. Now if the JUnit is integrated into the main build then once the deployment of sub projects has been finished, then all the builds will need to call the main build in order to get the unit test cases for the sub projects to be executed. This is required because every project needs to provide the path of its test individually to JUnit framework which executes the unit test cases of that project. This is not feasible since now after the deployment every build will call the build again, that will result in increase in the build deployment time and the time is a major issue.
- The second possibility is to integrate JUnit into all the build files individually so that there will be no need to pass the paths and other parameters from one file to another file. But this is also not feasible since this will result in redundancy of the code as well as increase in size of all the build files. If the JUnit is implemented in all the build files then the same code is added into all the files which is totally infeasible because the code should never be redundant in same project. Thus this method is also not suggested.
- The third possibility is to create a separate ant build file and add JUnit task to this build file and import that file into the builds at runtime and get the unit tests of all the projects executed. If this method is used then there is no need to call the main build file from all the builds since the new build file can be imported in all the sub-project builds. This will save the time resulting in decreased deployment time as well as it prevents code redundancy.

As per the given positive and negative aspects, the third method is best suitable to integrate Junit framework into the build. A separate .xml file is created which is only used to run the unit test cases. The implementation of unit testing framework in ant build is described with number of code snippets.

Every project has unit test cases which are kept in the test repository of the project, so the first and foremost condition is to check whether the project has test cases or not i.e. the project has test repository or not. The default target in runUnitTests.xml is *tests*, which first defines a condition property which checks if the project has test repository or not. The *tests* internally calls *runtests* target with parameters necessary for JUnit.

```
<target name="test">
    <condition property="${testDir.exists}">
      <available file="${test.dir}" type="dir"/>
    </condition>
  <antcall target="runtests">
      <param name="junit.format" value="xml" />
      <param name="junit.format.usefile" value="true" />
      <param name="junit.report" value="true" />
      <param name="haltonfailure" value="false" />
    </antcall>
  </target>
```

In *runtests* target first the existence of test repository is checked and based on that the target is called which performs compilation of .java files of unit test cases.

```
<target name="do-compile-tests" if="${testDir.exists}">
    <javac srcdir="${test.dir}" destdir="${test.build.dir}"classpathref="atl.test.class.path"
deprecation="false" source="1.7" debug="true" encoding="UTF-8"
includeAntRuntime="no"/>
</target>
```

The most important parameter here is *classpathref = atl.test.class.path* which defines the parameter which should be used by the system while executing unit test cases.

```
<path id="atl.test.class.path">
    <pathelement location = "${source.dir}/classes" />
    <pathelement path="${test.build.dir}" />
    <pathelement path="${test.dir}" />
    <pathelement path="${junit-jar}" />
    <pathelement path="${test.build.dir}" />
</path>
```

Now once the tests have been compiled successfully then the tests are needed to be executed by the JUnit framework which is finally finished by the *runtests* target in which the JUnit features are embedded. This target executes the unit test cases which were compiled by the *do-compile-tests* target. Basically what happens in *do-compile-tests target* is that the unit test cases are compiled and are converted to .class files and later these class files are executed by the JUnit libraries which are included into the ANT build using junit.jar file. This jar file path is provided using *atl.test.class.path* parameter. When the path of this jar file is passed into the build file then at the time of execution all the class files available in the jar files are extracted and are used to perform execution operation. The *runtests* target calls *failtests* and *reporttests* target. *Failtests* is used in case some unit test case gets failed because of some problem with code and *reporttests* is used to generate report for the executed unit test cases. The below snippet shows *runtests* target of build file.

```
<target name="runtests" if="${testDir.exists}">
    <antcall target="compile-tests" />
    <condition property="junit.format" value="plain">
        <not>
            <isset property="junit.format" />
        </not>
    </condition>
    <condition property="junit.format.usefile" value="plain">
        <not>
            <isset property="junit.format.usefile" />
        </not>
    </condition>
```

```
<condition property="junit.pattern" value="**/*Test.class">
    <not>
      <isset property="junit.pattern" />
    </not>
  </condition>
  <condition property="should.junit.report">
    <and>
      <equals arg1="${junit.format}" arg2="xml" />
      <istrue value="${junit.format.usefile}" />
      <istrue value="${junit.report}" />
    </and>
  </condition>
  <junit dir="${test.build.dir}" printsummary="yes" newenvironment="yes"
      failureproperty="has.junit.failed" haltonfailure="${haltonfailure}" forkmode="once">
    <classpath refid="atl.test.class.path" />
    <formatter type="${junit.format}" usefile="${junit.format.usefile}" />
      </fileset>
    </batchtest>
  </junit>
```

The *reporttests* target is called in order to generate unit test case execution report for executed test cases.

```
<target name="reporttests">
  <junitreport todir="${test.build.dir}">
    <fileset dir="${test.build.dir}" >
      <include name="TEST-*.xml"/>
    </fileset>
    <report format="frames" todir="${report.junit.dir}" />
  </junitreport>
</target>
```

This target generates test case execution report which is shown in figure 22. This report shows information about packages, classes and the also the test cases implemented in the classes. The time taken to execute the test cases is also published in the report.

Home

Packages

rwms
rwms.astar
rwms.events
rwms.geom

rwms.geom

Classes

GeometryTest
LocationTest
ReferencePointTest

Unit Test Results.

Designed for use with JUnit and Ant.

Class rwms.geom.GeometryTest

Name	Tests	Errors	Failures	Skipped	Time(s)	Time Stamp	Host
GeometryTest	20	0	0	0	0.029	2015-07-05T12:40:00	AMISHARS-IN

Tests

Name	Status	Type	Time(s)
equalsNullObjTest	Success		0.001
pathDifferenceDiffObjExcZTest	Success		0.000
getFacilityIdTest	Success		0.000
toMapTest	Success		0.000
getWidthTest	Success		0.001
equalsDiffClassObjTest	Success		0.001
equalsSameObjTest	Success		0.001
pathDifferenceDiffObjIncZTest	Success		0.000
getIdTest	Success		0.000
toStringTest	Success		0.000
getLengthTest	Success		0.001
pathDifferenceSameObjTest	Success		0.000
pathDifferenceDiffObjDelZTest	Success		0.000
getHeightTest	Success		0.000
getTypeTest	Success		0.001
hashCodeTest	Success		0.001
equalsDiffObjTest	Success		0.000
getXTest	Success		0.001
getYTest	Success		0.000
getZTest	Success		0.000

Figure 22: Report generated by the JUnit

Now finally the *failtests* target is called which is used if some unit test case is failed then the build deployment stops there and shows test case were failed.

```
<target name = "failtests" if="has.junit.failed" >
    <fail message="Junits failed. See Logs above (or generated above)" />
</target>
```

In this target the property *has.junit.failed* is enabled or disabled in *runtests* target based on the status of the unit test case being executed.

Thus using runUnitTests.xml file JUnit framework is integrated in the build process of RWMS. With the help of this file test cases are compiled, executed and the report is generated by JUnit.

4.3 Calling JUnit target from various builds in RWMS

Integrating JUnit framework in RWMS build alone is not enough. The runUnitTests.xml file needs to be imported and the *tests* targets needs to be called in all the build files of RWMS. So that the unit test cases of all the projects get executed. In all the build files the runUnitTests.xml file is imported at run time. Same syntax is followed in all the files. A target *callJunit* is created which imports the runUnitTests.xml file in the respective build file. The *basedir* is the parent directory of the build project whose build isbeing deployed currently. and the default target is test which is the first target of execution sequence in the

runUnitTests.xml file. The code snippet shows the callJunit target in all the build files.

```
<target name="callJUnit" if="${junit.report}">
    <ant antfile="../build/runUnitTests.xml" dir="${basedir}/../build" target="test"
inheritall="true" />
</target>
```

This target is implemented in all the build files and using this the unit test case execution process becomes automated.

The algorithm for automating the execution process of unit test cases is as following:

4.4 Algorithm

Input: Path location for all the unit test cases.

Output: JUnit unit test case execution report for automated test case execution.

Step 1. Complete build specification of the java project using ANT script.

Step 2. Integrate unit testing framework in the ANT build file by adding .jar file to the class path.

Step 3. Import that ANT file in all the build files of the project.

Step 4. Put all the unit test cases in the Test directories for respective projects.

Step 5. Pass location of Test directories of the projects as a parameter in the *atl.test.class.path*.

Step 6. Start deployment process using ANT build command *ant –f "build path"*

Step 7. If any test case fails then repeat from step 4 to step 6 else go to step 8.

Step 8. All the test cases are executed successfully, unit test case execution report is generated.

Step 9. EXIT.

4.5 Flow Chart for Automated Unit Testing Process

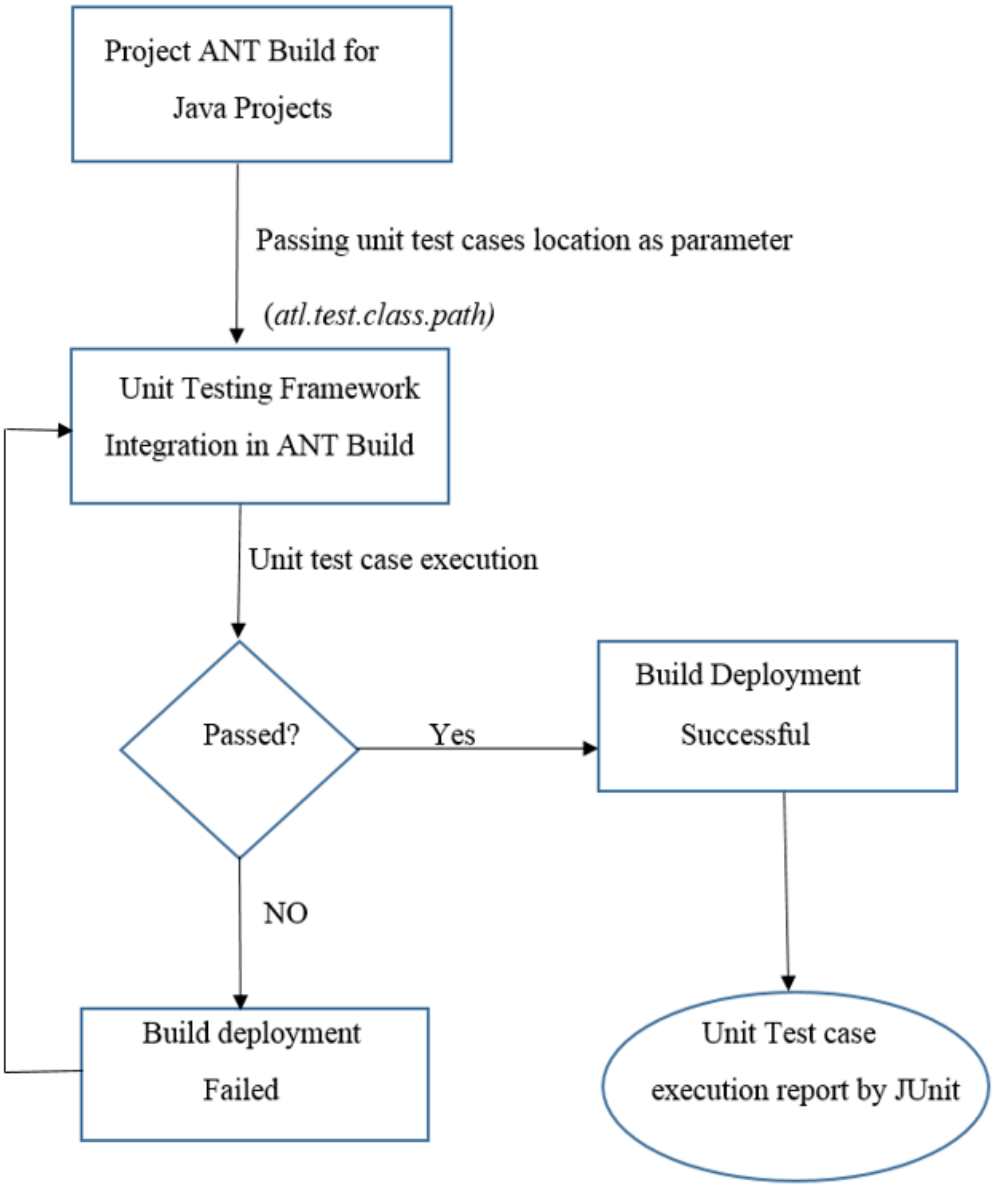


Figure 23 Flow chart for automated execution of unit test cases

Chapter 5 Results and Discussion

It is clear that with the increase in the complexity of the code, unit testing has become of prime importance. Unit testing helps to reduce the error and defects in the code making it long lasting and increasing the reliability of the code. Unit testing not only reduces defects but also it detects the error at very beginning or micro level so it becomes very easy to detect them and trace back the source of the error and to resolve it.

Now since the code base of projects has become very large so it is not feasible to execute all the unit test cases manually. This require a lots of human effort as well as it's a time consuming process. So the algorithm suggested in this thesis will help to not only reduce the human effort and time in executing the unit test case but using this method the execution of unit tests become quite easy and convenient process. Once the unit testing framework has been integrated in the ANT build of the project, it can be used any number of time. We just need to pass the path of the location of test directories of the projects where the unit test cases are placed. So this implementation is like install once, use forever kind of thing. Now the build, in which the unit testing framework is integrated, can be deployed on local environment as well as on servers where the build execution tools are integrated like Hudson etc. In local environment the deployment does not get completed because the process requires some resources which are only available only on the servers. However the process is not completed and the final result is failed but still all the operations which should be performed during deployment are performed completely and all the software artifacts, logs and static analysis reports and other reports are also generated in the way they are supposed to be generated only the final status of the result is failed which can be analyzed using the build logs. In order to locally execute the build of the java projects the system should have ANT installed and all the environment variables like ANT_HOME and JAVA_HOME should be defined properly. Once all the setup has taken place the deployment can be started with the Ant execution command using command prompt. The figure 24 shows build deployment in local environment. Once the build execution starts, if the unit testing framework is integrated into the build of the project then all the unit test case will get

executed automatically and the report for unit test case execution will be generated. All the reports and artifacts are available locally only since the deployment takes place on local environment.

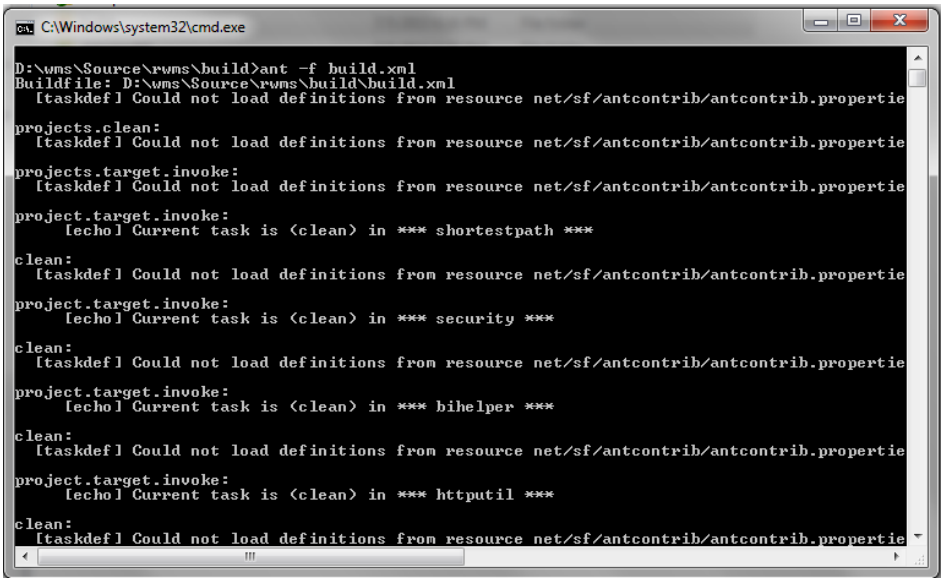


Figure 24: Build deployment in local environment

Now if the build is executed on the server then one should make sure that the whole code base is properly checked-in on the server and all the build resources are available. If not the case then the deployment will be unsuccessful otherwise the build deployment will be completed successfully and all the artifacts are stored on the server.

Figure 25: Build deployment on Hudson server

Figure 25 shows the successful build deployment on Hudson server. This shows overall time taken in build deployment. This time also includes time taken to execute the unit test cases of the project. In order to test how much time does it takes extra to execute unit test cases along with the deployment of other components, let us examine some test case execution scenarios. Below test case execution reports for three packages are shown.

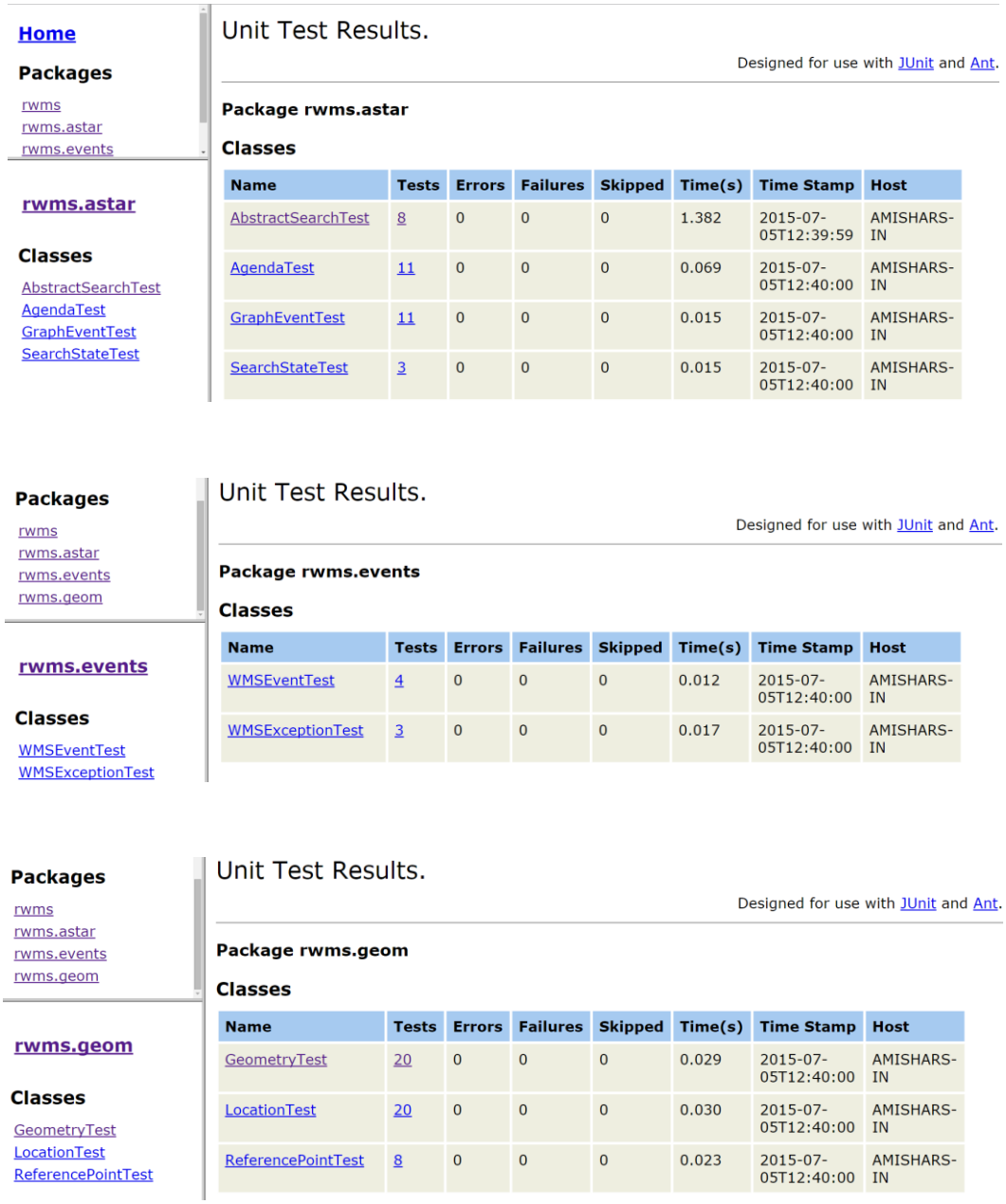


Figure 26: Unit test case execution report for various packages in RWMS

The figure 26 shows unit test case execution report for three projects. This shows all the classes available in a package, along with the information like number of test cases present in the class and then the time taken in executing the test cases of that class. For

a single test case the execution time ranges from 0.010 second to some 0.030 seconds. So we will take mean value which is 0.020 seconds taken to execute a single test case. There are 350 tests executed in the RWMS project. So time taken to execute test cases will be 350 * 0.020 = 7.000 seconds. If we increase the value of test cases to 1000 test cases then the total time taken will be approximately 20 seconds which is a very minor fraction of the time taken to execute the build which is 10 minutes.

Thus the final result is that even in execution of 1000 test cases the time taken is 20 seconds only. Thus even if the unit test cases are executed with the deployment then it does not create overhead in deployment process since the overall time taken to execute the unit test cases is a very small fraction of the total deployment time. This also saves the human efforts since the user just need to kick off the build and all the unit test cases of the project gets executed automatically without any human intervention required.

Comparing with manually executing the unit test cases in project the automation process is a lot faster and imposes less overhead. Talking about manual execution, developer needs to manually execute all the test cases one by one. Suppose if the developer takes almost 30 to 50 seconds in order to execute the test case, since first he/she needs to go to individual test case and then execute it. However the time taken in executing the test case is similar to automated process i.e. average 0.020 seconds but the overhead in this process is very high since the developer will need to access all the test cases and then execute them. Since the efficiency of human varies from person to person, so it may be the case that the person is not that efficient in working, then it will take very long time. If a single unit test case require average 40 seconds in completing the execution process then in executing the 1000 test cases it will be 40 * 1000 = 40,000 seconds which is approximately 11 hours work and adding all the factors the time becomes very high.

So the automated process results show it is many times faster than the manual process with a very little overhead.

Chapter 6 Conclusion and Future Scope

It can be concluded from above statistics provided for the unit test case execution that the time required only for the execution process is very small. This time is only in the fractions of a second, it is the process adopted to execute the unit test cases of the project which defines the total time required to execute all the unit test cases. If the process is manual i.e. all the test cases are executed by the developer himself then there are lots of overhead included in the execution process which adds a lot of time in software development life cycle. This requires very high amount of time and human effort which affects the quality of the development process. This may even cause the product to not meet its requirements.

The automated process of unit testing execution using ANT build of the project is the answer to all the overheads which are there in manually execution process. This reduces the execution time required to execute all the unit test cases up to such small value that it is negligible as compared to the manual process of executing the test cases. This makes the process free from human intervention removing the chances of human errors which may include like some test cases may have left unexecuted or there may some errors exist which might have left unnoticed. Thus not only the time and human efforts overhead has been reduced but also the quality of the development process is improved. Thus this provides improvisation in the software development process quality which is the ultimate goal of the SDLC.

The future scope of automated unit testing is very wide. Now in software development organizations all the processes are being converted into automated ones like automation testing is the hottest development in the testing process which saves you from executing manual test cases one by one just by executing the scripts. When all the processes are going automated way then why not unit testing! In forthcoming future the complexity of software will be very high. Unit testing has always been the best practice to test the efficiency of the code and when the code will become more complex, its importance will also grow high. This automation process will allow the developers to write more and more number of test case to make sure there are no vulnerabilities in the code without worrying about the overhead in execution of the unit test cases. Thus it will not only increase the process quality but also make the developers code with more confidence resulting in a better reliable product.

References

[1] IEEE, "*IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminoingy*," , 1990.

[2] Seema and Sona Malhotra, "*Analysis and tabular comparison of popular SDLC models*," International Journal of Advances In Computing and Information Technology, vol. 1, issue 3, pp. 277-288, 2012.

[3] Shikha maheshwari and Dinesh Ch. Jain , "*A Comparative Analysis of Different types of Models in Software Development Life Cycle*," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, issue 5, May 2012.

[4] Noor, N.L.M., Adnan and Mansor S., "*A survey on user involvement in software Development Life Cycle from practitioner's perspectives*," Computer Sciences and Convergence Information Technology (ICCIT), pp. 240-243, 2010.

[5] Roger S. Pressman, "*Software Engineering: A Practitioner's Approach*," 5th Edition, New York: McGraw-Hill Higher Education.

[6] T Bhuvaneswari and S Prabaharan, "*A Survey on Software Development Life Cycle Models*," International Journal of Computer Science and Mobile Computing, vol. 2, issue 5, pp. 262-267, 2013.

[7] Rodriguez Martinez and Alvarez F.J., "*A Descriptive/Comparative Study of the Evolution of Process Models of Software Development Life Cycles* ," Computer Science (ENC), Mexican International Conference, pp. 298-303, 2009.

[8] P.K.Ragunath, S.Velmourougan, P. Davachelvan, S.Kayalvizhi and R.Ravimohan, "*Evolving A New Model (SDLC Model-2010) For Software Development Life Cycle (SDLC)*," International Journal of Computer Science and Network Security, vol.10, no.1, 2010.

[9] Shivkumar Hasmukhrai Trivedi, "*Software Testing Techniques*", International journal of Advanced Research in Computer Science and Software Engineering, vol. 2, issue 10, 2012.

[10] Yoonsik Cheon and Gary T. Leavens, "*A Simple and Practical Approach to Unit Testing: The JML and JUnit Way*," Computer Science Technical Reports, Iowa State University Digital Library, vol. 2374, pp. 231-255, 2002.

[11] Nachiappan Nagappan, E. Michael Maximilien, Thirumalesh Bhat and Laurie Williams, "*Realizing quality improvement through test driven development: results and experiences of four industrialteams*," Empirical Software Engineering Journal, vol. 13, issue 3, pp. 289-302, 2008.

[12] Aleksandar Bulajic, Samuel Sambasivam and Radoslav Stojic, "*Overview of the Test Driven Development Research Projects and Experiments*," In Proceedings of Informing Science & IT Education Conference (InSITE) 2012,

[13] Beizer B. ,"*Black Box Testing: Techniques for Functional Testing of Software and Systems*", Software, IEEE vol. 13 , issue 5, Sept. 1996.

[14] Rankin C, "*The Software Testing Automation Framework*", IBM Systems Journal , vol. 41, issue 1, 2002.

[15] Andrew Hunt and David Thomas (2003),"*Pragmatic unit testing in Java with junit: The Pragmatic Starter Kit – Volume II*," Texas: The Pragmatic Bookshelf.

[16] Hashish N. and Bottaci L. "*Language Constructs for Generalising Unit Tests*", Testing: Academic and Industrial Conference - Practice and Research Techniques, pp. 129-130, 2009.

[17] Cendric Beust, Hani Suleiman and Brian Goetz (2008), "*Next Generation Java Testing: TestNG and Advanced Concepts*," Boston: Pearson Education.

[18] Khalid Almerien and Kenneth Magel, "*Examining the Effectiveness of Testing Coverage Tools: An Empirical Study*," International journal of Software Engineering and its Applications,vol.8, no.5, pp. 139-162, 2014.

[19] Ben Chelf, "*Software Build Analysis: Eliminate Production Problems to Accelerate Development*," International Journal of Computer Science and Network Security, vol.10, no.1, January 2010.

[20] D. Gelperin, B. Hetzel, "*The Growth of Software Testing*," CACM, vol. 31, no. 6, 1988.

[21] IEEE, "*IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology*," (Revision and redesignation of IEEE Std. 729-1983), Institute of Electrical & Electronics Engineers and the American National Standards Institute, 1991.

[22] Wirth N.,"*A Brief History of Software Engineering*," Annals of the History of Computing, IEEE vol. 30, issue 3, pp. 32-39, 2008.

[23] Kent Beck, "*Extreme programming explained: embrace change*," Boston: Addison-Wesley Longman Publishing Co., 5th edition, 1999.

[24] Laurie Williams, Gunnar Kudrjavets, and Nachiappan Nagappan, "*On the Effectiveness of Unit Test Automation at Microsoft*," IEEE Conference on Software Reliability Engineering, pp. 81-89, 2009.

[25] Andrew Patterson, Michael Kolling and John Rosenberg, "*Introducing Unit Testing with BlueJ*," ITiCSE '03 Proceedings of the 8th annual conference on innovation and technology in computer science education, pp. 11-15.

[26] Ravinder Kumar and Mr. Karambir Singh, "*Enhancing Component Based Testing Using JUnit Tool in Net Beans Environment*" , International Journal of Advanced Research in Computer Science and Software Engineering, vol.2, issue 7, July 2012.

[27] Emad Shihab, Zhen Ming Jiang, Bram Adams, Ahmed E.Hassan and Robert Bowerman,"*Prioritizing the Creation of Unit Tests in Legacy Software Systems*," Software – Practise and Engineering, Published Online in Wiley InterScience, 2010.

[28] Fraser, G. and Daka, E,"*A Survey on Unit Testing Practices and Problems*," Software Reliability Engineering (ISSRE), IEEE 25th International Symposium, pp. 201-211, 2014.

Publications

Research Paper Communicated

Amit Sharma, Shivani Goel, "*Comparative Analysis of Unit Testing Frameworks for Java projects*", Communicated to NGCT 2015.

Acceptance Date – 30th July, 2015.

Video Presentation Link

http://youtu.be/OX5U1oHmUSU

Reflective diary shows the reflection of experieces, a person have during his journey of learnings. Here I have tried to recollect all my learning experiences which I had during the one year journey of my internship. I have also presented the experiences I had while working on my thesis.

June, 2014

My internship with Oracle Retail started on 2nd June, 2014. This is always amazing experience when you start working in an industrial environment just after your college life. The first month had lots of new experiences as it was my first experience in a software organization. Initial 10 days I was given the introduction by Human Resource people that what process is followed in the organization. Introduction of the product which are developed at Oracle Retail was given. Product based trainings were given which included short demos of all the products developed in the organizations. These trainings were really full of knowledge as the demos were given by the experts of that departments. After that technical trainings phase started. I attended many technical trainings which included Java, Java Script, PLSql, Open Script and Oracle Fusion Middleware training. These trainings were very short term as compared to the scope of the content but still the knowledge conveyed in these were really helpful. So my first month was lots of learnings and getting familiar with how the process works in the software industry.

July, 2014

Till 15th July my technical trainings were going on which were started in June. After 15th July I was assigned in Retail Warehouse Management System (RWMS) team. The main issue I came to know that I was the only person working in India in RWMS product. So initially setup was bit complex. Just after getting assigned to RWMS team, I was given two weeks training about RWMS. I learnt many aspects of this product. These training were organized to make me familiar with the product. I learnt what are the basic functionalities of RWMS? What work is going on with the product? What features have been included and what are the plans for future? I learnt about the

frameworks which are used to develop RWMS. This training was completed in the end of the July month. I was also given some documents which included product specification and requirements for the future plans. After the self-study of the documentation and trainings, now I had a technical and functional knowledge of the products on which I am going to work.

August, 2014

My first two months at Oracle went in attending various trainings only. In August I started to work with RWMS product. I completed the setup which was required for RWMS. I setup the application in my local machine. I learned the procedures, standards and the development life cycle followed by my team. I was supposed to complete tasks assigned to me and report the status of the tasks to the whole team in weekly status meetings. My first task was related with deployment of the product. This was completely new thing for me because the deployment is done using build files of the product. These build files are written using ANT script. I went through online tutorials of ANT script and also did a lot of research about how to develop builds using ANT script.

My first task was to optimize the deployment time of RWMS. At this time the build size was very large and the time taken in deployment was high. I went through documents of various products having similar task flows. I got to know the size of build can be reduced by removing the redundant code and the deployment time can be reduced by removing the dependencies among various task. This task was completed by me in given time schedule.

This month I learnt ANT script which is used for the deployment of the build files and I also learnt to develop build files for the product. I also learnt the work flow which is followed in my team.

September, 2014

Up to this month I have become completely familiar with my work and what I needed to keep learning in order to provide my work within given time schedule. At the starting of first week of August I was given the task of making the build hierarchy free from user intervention. This was a task which included study of dynamic systems. Since in order to remove the user dependency the required properties were needed to be defined

at run time while the deployment is in progress. I had to go through documents of dynamic systems. I learnt dynamic features of ANT script which included sync of java code with ANT script. I learnt to deploy the product on server and also on my local environment. I started doing the testing for various features of product after deployment to check whether the newly added changes are working fine or not. I learnt all these by going through online tutorials and also internal documentation.

As the work load increased the main thing I had to face was that I was the only person here in my team. If I needed some help I had to get the work done with the help of members of some other teams who had knowledge related to my domain. So that is how I learnt to get my work around with help of different teams. So in September I learnt dynamic concepts of deployment.

October, 2014

This month I got my task which was related to development. I had to enhance the functionalities of the deployment of the product. My task was to add static code analysis tools in the build of the RWMS. The aim of the task was to perform static code analysis of the RWMS code at the time of deployment, so that if there are any errors exist which may become defects in later stages. I studied about the functionalities of various static analysis tools like checkstyle, findbugs etc. I went through the research done in this field and learnt how the code analysis takes place. Then I learnt to integrate the tools in ANT script. I made the changes in code and checked in the code on the server.

One incident I remember about this task is that after checking in I had to deploy the product on Hudson server. First I tested the build deployment on my local machine and after successful testing I checked in all the files which I changed. Once the checking in of files was completed I had to deploy the product on Hudson server. The tools like checkstyle and findbugs are stored in a repository called JavaTools on server. The build file needs to access these files in order to complete the static code analysis. The check in of tools was done recently on the server by me. At the time of deployment I accidently forgot to define the path in the externals of the server which caused all the java tools to be extracted in wrong directory at run time and the build deployment was failed. Now to recover from this situation I had to check in all the java tools again on the server which was a very time consuming task and the final deployment was delayed by one day from the schedule. This incident taught me to follow all the steps carefully and

from that day onwards I started making list of all the steps I need to follow and then performed all the final execution task.

November, 2014

Once the integration of static code analysis tools was completed, I was deployed in new task other than the build files. My next task was implementing unit testing for the RWMS project. For this I was given longer time line. First I had to do the code review of the java files for which I needed to write unit test cases. I started with first reading about the tools which are used for implementing the unit test cases for java projects. I learnt about JUnit tool. Then I started going for tutorials of JUnit. I learnt to write various JUnit asserts and methods. After learning about unit testing tools I started working with the code base of RWMS. I had multiple KTE (Knowledge Transfer) with the developers of my team who developed the code. This KTE helped me to understand the logic which was implemented in that code. After that I started writing small unit test cases for simple java classes. This helped me to get started with unit testing. So this month I began my journey with unit testing. I learnt basic functionalities of unit testing tools. I started working with JDeveloper IDE which is the tool used to develop java based projects.

December, 2014

I continued working with unit testing of RWMS. I also implemented unit tests for components which is little bit complex to understand and requires some time to implement. This was the month when I got the topic for my thesis. This happened because of a task assigned to me to automate the execution of unit test cases for RWMS. At one side I was implementing unit test cases and on other side I was assigned the task to look for a solution to automatically executing the unit test cases. The plan was to have this functionality in the release which was going to happen in April, 2015. The requirement came because of the increasing number of unit test cases. I started with the research in this area related to automated unit testing. I went through research papers which gave me idea what processes have already been implemented to automate the unit test cases. This month I prepared my problem statement and got it approved by my mentor, so that I could start working on it from the perspective of thesis.

January, 2015

I got to know similar framework is being used by some other project at Oracle. I went through the documentation of that project but I found out that logic could not be implement in RWMS as that project was implemented in Oracle forms while RWMS is developed in ADF and pure java environment. I looked for previous research again in order to see which tool could be used to make this process automated.

I got finally answer to my question that JUnit tool could be used to make this process automated. This answer I found out when I was going through one of my deployment documents which mentioned the logic of various tools can be integrated into the ANT build of the project. I discussed this with my team and they approved the idea. The problem here they mentioned was that this logic has never been implemented in the Oracle products. So there were no internal documents available. I was given two weeks' time to look around for possible solution. I started working with JUnit again.

This month I got the answer to my question that which tool will resolve the problem to automate the unit testing of RWMS.

February, 2015

I started reading the research papers and various content available on the internet related to JUnit. I started going through documentation of JUnit which is an open source framework to implement the unit testing of java projects. There is a plugin available to integrate the JUnit into the IDE software which is used to develop the code base of the product. The problem was that no plugin is available to integrate into the ANT build of the java project. What solution I found is that I could learn how the test cases are executed by the JUnit framework and then I could integrate the logic into the ANT build of the RWMS. I read the research papers to see how could I learn the execution logic of the JUnit. I read a book on JUnit.

Finally I came to know that JUnit.jar file is stored on the system. This jar file is added into the application libraries and class path. I used that jar file and extracted it. I analyzed the byte code packaged in the Junit.jar file. I learnt the whole code of JUnit which was executing the unit test cases.

March, 2015

This month I started implementing the logic of JUnit test cases execution logic into the ANT file. Now the problem was I had to integrate it in a way that it should not increase the deployment time which would be infeasible. Also I needed to make sure that all the execution should be automated only, there should be no human intervention required. First I implemented my first logic in which I integrated the JUnit logic in all the build files of RWMS. This worked and the test cases were executed successfully but what review I got from my code reviewer is that the implemented code was redundant. Redundant code is never used in the project. However I got the desired output but it was not feasible and was rejected. The next solution I came up with was to implement the JUnit logic in the main build file of RWMS but that was resulting in increased deployment time as there were already very high number of operation defined in the main build file. So this solution was also rejected.

Finally after researching about the deployment process of java products and studying more about ANT build script. I suggested my team about implementing the logic of JUnit in separate file and then import the file in all the build files at run time. My team approved this and I tried implementing that logic. I integrated the JUnit logic in a separate file and deployed the project. The result was successful automation of unit test cases of RWMS and also it did not increase the deployment time of the project. My team reviewed my code and it was approved by development team that it won't affect the performance of the product and it also provides the required output.

This month I have successfully implemented the framework I needed to implement and it was also working fine. So 80% work of my thesis was completed by this month.

April, 2015

The framework has already been implemented in March and now I was ask to develop the algorithm so that could be documented into the internal docs of RWMS. I worked on all the basic steps and prepared the document which could be used in future to automate the unit testing process of any java project. I also prepared the system flow diagram of the implemented framework. In the mid of April I was given the task to compare the results of this framework with manual execution of test cases. I performed various test cases by deploying the product many times and also executing the unit test

cases manually one by one. This took me 10 days to come up with the comparison results and I sent it out to my team which was appreciated. Once the framework has been tested completely, it was integrated in the version which was released in first week of May.

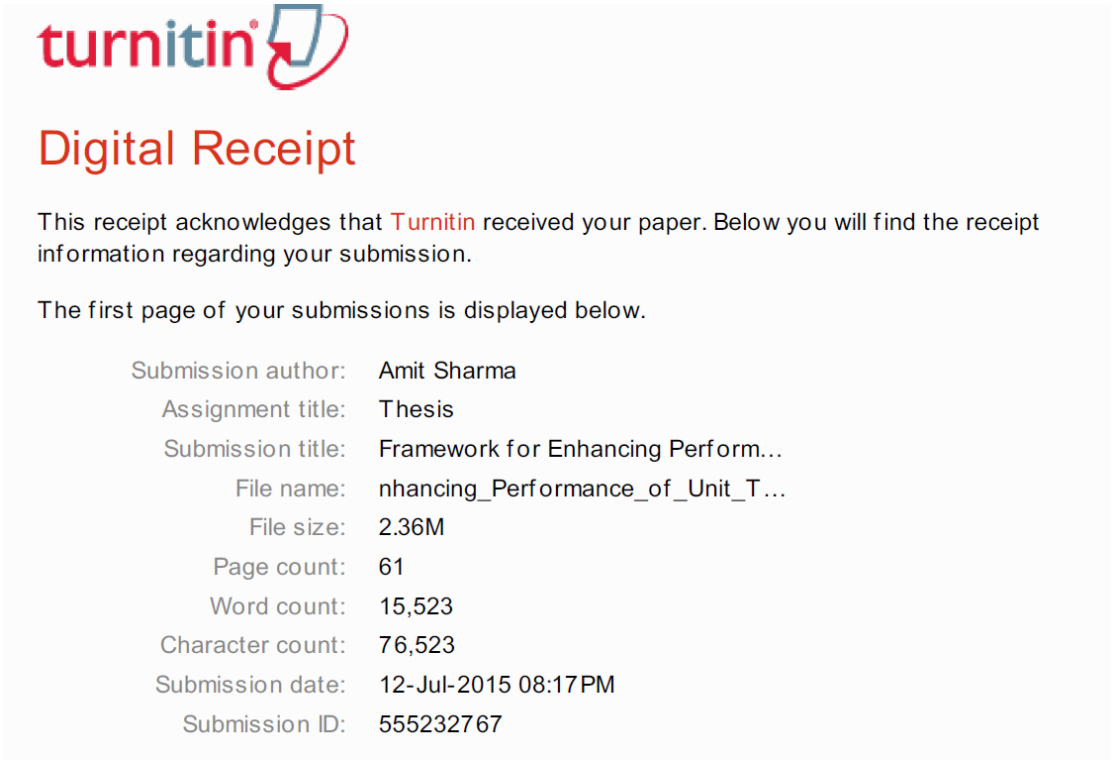
I was given Oracle ExaThanks certificate which is given by the manager as a sign of appraisal of my contribution to the product.

In the last week of April as the development work of RWMS was completed so I was moved to automation team of RWMS. This team used to perform the automation testing of the RWMS. The tool used was open script. I had gone through a training for Open Script in my first month of internship. I was familiar with the tool so I started working with the automation testing of RWMS.

May, 2015

In this month I started writing my thesis as I had successfully implemented the framework and I was able to come up with a good algorithm. I was also working in automation testing of RWMS. I learnt to write automated test scripts using Open Script. I was also going through documentation of Advanced Clustering which is a product of Analytics department. I was reading the documents since I was scheduled to start working for that team from the third week of May. In the second month the automation work of RWMS was finished and I started working with analytics team. I went through number of KTEs by the team members of analytics and setup the application on my local environment. By the end of May the setup was completed and I started learning about the code base of the Advanced Clustering.

So this was a journey with lots of learnings. I learnt how to work in industrial environment. I learnt things which I will keep using forever in my career. I learnt new technologies. I worked with new products. I worked in multiple domains which included development, testing and deployment. This gave me experience of multiple domains. So this was a lifetime experience for me.



turnitin

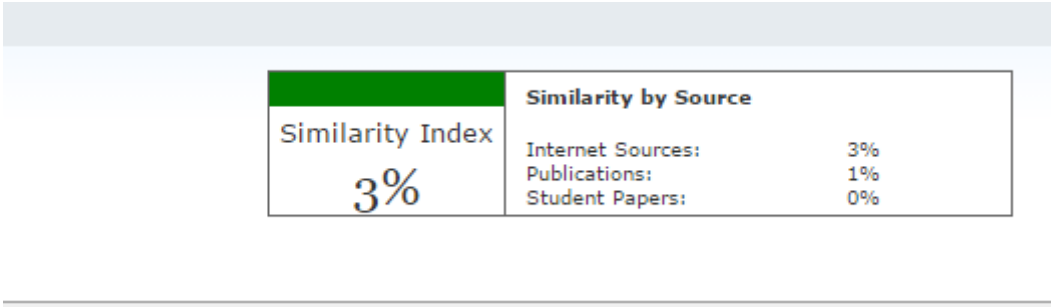
Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Amit Sharma
Assignment title:	Thesis
Submission title:	Framework for Enhancing Perform...
File name:	nhancing_Performance_of_Unit_T...
File size:	2.36M
Page count:	61
Word count:	15,523
Character count:	76,523
Submission date:	12-Jul-2015 08:17PM
Submission ID:	555232767

Figure. Digital Receipt generated by Turnitin



Similarity Index	Similarity by Source	
3%	Internet Sources:	3%
	Publications:	1%
	Student Papers:	0%

Figure. Plagiarism report generated by Turnitin