International Journal of Civil Engineering and Technology (IJCIET)
Volume 9, Issue 1, January 2018, pp. 944–952, Article ID: IJCIET_09_01_093
Available online at http://http://www.iaeme.com/ijciet/issues.asp?JType=IJCIET&VType=9&IType=1
ISSN Print: 0976-6308 and ISSN Online: 0976-6316

© **IAEME** Publication **Scopus** Indexed

COMPARISON OF TDD AND PAIR PROGRAMMING FOR IMPROVING SOFTWARE QUALITY

Sushma VS and Nizar Banu PK
Department of Computer Science, CHRIST (Deemed to be University),
Bengaluru, India

ABSTRACT

These days, programming improvement groups utilizing coordinated procedures have started widely adopting Test-driven development and Pair Programming. Test-driven development (TDD) is a transformative way to deal with improvement, which joins test-first improvement where you compose a test before you compose simple enough creation code to satisfy that test and refactoring. Pair Programming is a sort of communitarian programming where two individuals are working at the same time on a similar programming task. In this paper the TDD and Pair Programming is applied for a dataset, collected from a group of users and compared. For our research, we executed structured experiments with five set of pair programmers and ten individual programmers. Both groups developed programs in Java. The outcome acquired demonstrates the strategy helps in expanding the software quality.

Key words: Test-Driven Development, Pair Programming, Software quality

Cite this Article: Sushma VS and Nizar Banu PK, Comparison of TDD and Pair Programming for Improving Software Quality. *International Journal of Civil Engineering and Technology*, 9(1), 2018, pp. 944-952.
http://www.iaeme.com/IJCIET/issues.asp?JType=IJCIET&VType=9&IType=1

1. INTRODUCTION

AGILE has become today's buzzword when describing a modern software process. AGILE methodology is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. Both development and testing exercises are simultaneous. Agile teams, committed to frequent, normal, high quality production, find themselves struggle to find ways to keep short-term and long-term efficiency as high as possible. The most popular agile methodologies include Extreme Programming (XP), Scrum Crystal, Test driven development, Pair Programming.

Pair programming (pp) comprises of two software engineers sharing a solitary workstation (one screen, console and mouse among the match). The developer at the console is generally called "driver", the other, likewise effectively associated with the programming undertaking but focusing more on overall direction is the "Navigator", it is expected that the programmers swap roles every few minutes(creator becomes quality assurer and vice versa.) [8]. It requires

the two software engineers to have the vital delicate aptitudes required for joint effort and the important hard abilities to compose and test code. The name of the procedure, pair programming can lead individuals to erroneously expect that programming engineers just match amid code advancement. Nonetheless, blending can happen amid all periods of the advancement procedure, in combine configuration, match troubleshooting, match testing, etc. Software engineers could combine up whenever amid advancement, specifically when they are working on an unpredictable or new issue. Combine writing computer programs is additionally a viable approach to circulate space and framework information all through the group. Advocates of Pair programming guarantee that it supports long-term profitability by significantly enhancing the nature of the code. In any case, any reasonable person would agree that for various reasons, matching is by a long shot the most dubious and slightest generally grasped of the spry developer hones. The hypothesis is that blending brings about better plans, less bugs, good value and much satisfactory spread of information over an advancement group, and therefore greater usefulness per unit time, measured long haul [4].

"**Test-driven development**" refers to a style of programming in which three exercises are firmly joined: coding, testing (through composing unit tests) and outline (through refactoring).TDD also known by the names as, Test First Design (TFD), Test First Programming (TFP), Test Driven Design (TDD). Today broadly received in industry both as a major aspect of a vast scale and as remain stand-alone practice. TDD is generally thought to be the amalgamation of test-first improvement, in which unit tests are composed before the usage code expected to finish those tests and refactoring, Which incorporates rebuilding a bit of code that passes the tests keeping in mind the end goal to lessen its unpredictability and enhance its clearness, understandability, extendibility, as well as viability. TDD is frequently portrayed with the alleged "redgreen- refactor cycle" [6].

1.1. Types of TDD

There are two types of Test Driven Development such as Acceptance TDD and Developer TDD, which explained in the following sections

- Acceptance TDD (ATDD): With ATDD, user can write a single acceptance test. The objective of ATDD is to specify detailed, executable requirements for your solution on a just in time (JIT) basis [12].
- 2) Developer TDD**:** With developer TDD user, compose a solitary developer test. The objective of engineer an advancement group, and therefore greater usefulness per unit time, measured long haul [4].

1.2. Test Structure

Successful format of a test case guarantees every single required activity is finished, enhances the decipherability of the experiment and smooths the stream of execution. Steady structure helps in building a self-reporting experiment. A regularly connected structure for test has four stages as shown in the FIGURE 1.

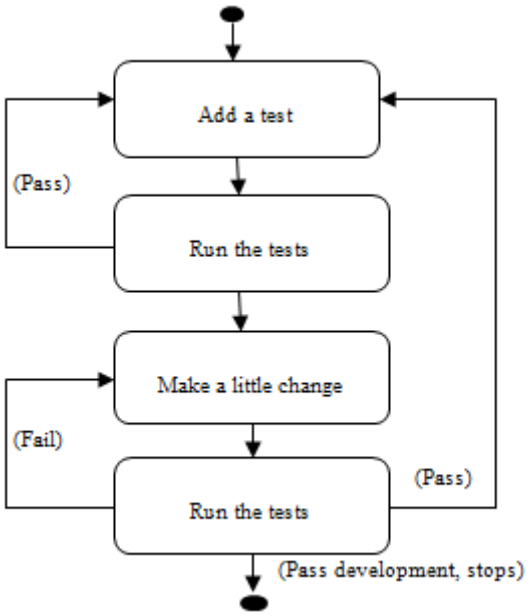

Figure 1 Shows the Test Structure of TDD

- **Add a test**: Put the Unit Under Test (UUT) or the general test frame work in the state expected to run the test.
- **Run the test**: Trigger/drive the UUT to perform the target performance and capture all yields, such as return values and output parameters. This progression is generally extremely straight forward.
- **Make a little change**: Ensure the aftereffects of the test are right. These outcomes may incorporate express yields caught during execution or state changes in the UUT.
- **Run Tests**: Restore the UUT or the general test framework to the pre-test state. This restoration allows another test to execute quickly after this one [9].

1.3. Limitations

Test-driven advancement does not perform adequate testing in circumstances where full functional tests are required to decide achievement or disappointment, because of broad utilization of unit tests. A high number of passing through unit tests may bring a misguided feeling that all is well and good, bringing about less extra programming testing exercises, for example, integration testing and compliance testing. Composing and maintaining an unreasonable number of tests costs time [1].

1.4. Applications

The end imperfection content is exactly lower. The plans are better and code length shorter. Individuals figure out how to cooperate and talk all the more regularly together, giving better data stream and group elements. The group tackles issues quicker. Individuals learn altogether more about the framework and about programming improvement. Test-driven improvement can deliver uses of high caliber in less time than is conceivable with more seasoned strategies. Proper implementation of TDD requires the developers and to properly understand how the application and its highlights will be utilized as a part of this present reality [11].

2. BACKGROUND

We present and discuss experiences from literature related to the case study that includes TDD and PP methodologies.

The research in [4] mainly concentrated on the pair programming, Pair programming is flair of programming in which two programmers work together at one computer consistently teaming up on a similar outline, calculation, code, or test. They used the pair programming to increase the product quality, increase the team soul and help in learning the administration. It helps in industry, schools, colleges etc. In this they did research on the school students. They conducted experiment on how pair programming helps the students to increase the student's spirit, understudies to be more effective, and enhances understudy maintenance in a data innovation. The research in [3] mainly concentrates on how and where the pair programing is used. The enthusiasm for pair programming (PP) has expanded. Though, numerous items of common sense of PP are ineffectively caught on. We exhibit encounters of utilizing PP widely in a modern task. The reality that the group had a predetermined number of top of the line workstations constrained it decidedly too snappy sending what's more, thorough utilization of PP. The engineers preferred PP and learned it effectively. At first, the sets were not turned as often as possible however embracing day by day, arbitrary revolution made strides the circumstance. Visit turn appeared to move forward information exchange. The driver/pilot parts were exchanged from time to time, yet the accomplices conveyed effectively. The pilot seldom spotted imperfections amid coding, yet the discharged code contained nearly no imperfections. Test-driven development and planning combines conceivably diminished imperfections. The designers considered that PP enhanced quality and information exchange, what's more, was more qualified for complex assignments than for simple errands. The research in [5] mainly concentrated on distinguishing between the pair programmers and individual programmers. The senior programming student's participated in an organized test. Its motivation was to approve quantitatively the narrative and qualitative pair programming results observed in industry.

Example: The 85% of students verified a preference for pair programming. A while later, a large number of the 35 conceded that they were at first hesitant, yet inquisitive, about Pair Programming. They separated the students into two groups, both made out of a similar blend of high, normal, furthermore, low entertainers, as dictated by their review point midpoints. Both the pair programmers and individuals have coded the similar programs but the overall performance has been changed between the two groups. Finally the pair programming and individual's results are presented Figure2

List of Programs	Individuals	Pairs
Program1	73.4%	86.4%
Program2	78.1%	88.6%
Program3	70.4%	87.1%

Figure 2 Shows the percentage of test cases passed

3. EXPERIMENTAL SETUP

AGILE methodology describes a set of values and principles for continuous iteration of improvement and testing throughout the software development lifecycle of the project. The famous agile methodologies include Pair Programming, Test driven development Etc. Pair programming is an agile software development comprises of two programmers cooperate at one workstation (one screen, console and mouse among the match). It is expected that the programmers swap roles every few minutes. Test-driven development is a software improvement process that depends on the reiteration of short development cycle: refers to a style of programming in which three exercises are firmly joined: coding, testing and design. The degree of interdependence between two modules is known as "Coupling Factor". We aim to reduce coupling- to make modules as autonomous as possible. The frame work followed in this paper is shown in Figure 3 and described below

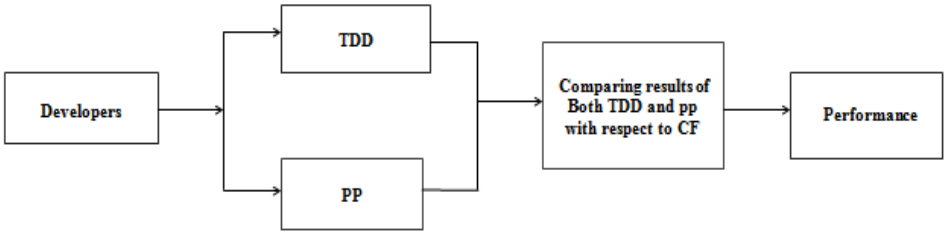

Figure 3 Shows the overview of complete work

Step1: Initially the set of developers are gathered.

Step2: The programmers are allocated into two groups. The first group developers follow the Test-Driven Development and the second group programmers follow the pair programming methodologies.

Step3: The results of step two are compared. The comparison results analyzed on the basis of Errors, LOC, Time consumption, Number of times debugging and coupling factor.

Step4: The outcome is based on results of third step. Finally results depend on the performance of agile methodologies (TDD and PP).

4. RESULTS AND DISCUSSIONS

In this paper, the set of planned experiments were executed by 20 professional programmers, 20 programmers are separated into two groups each group contains 10 individuals. One group builds up the code utilizing TDD and other gathering builds up the code utilizing Pair Programming. In Pair Programming the 10 individuals are gathered as pairs, it contains 5 pairs. Both the groups developed programs in Java. After the fulfilment of the execution we have looked at the LOC, Time, Variable-Declaration, No of times they debugged, errors and coupling factor. Later we looked the result of all the key-words and calculated coupling factor of the both groups. We found that pair programmers built-up a program with higher quality, error free and they developed the code within the time. Where the TDD programmers built-up the program with good quality, yet not error free and further more they debugged the code more than the pair programmers and they have taken additional time than pair programmers. At last we found that coupling factor is high in the TDD programmers than the Pair programmers. When the coupling factor is high the quality will be less in the TDD than the pair programming.

4.1. Pair Programming

For Pair Programming five programs which include the concepts like multiple-inheritance, sum of 'n' numbers, a package for sorting with selection and insertion sort, reversing a number, transpose of matrix and displaying the employee details are given. The following program is the primary program of a given set. It is developed by the pair programmers' in which they have a chosen Transportation as a super class, Vehicle is a sub class of super class and car is the sub class of Vehicle. Where each method is not much dependent to one another, if interdependency is less coupling will be less and then the quality will be high.

```
import java.util.Scanner;
class transportation
{   int price;
String color;
Scanner s=new Scanner(System.in);
transportation()
{color="";
price=100000;}
transportation(String color)
{ this.color=color;}
transportation(String color,int price)
{ this(color);
this.price=price;}
void read()
{ System.out.println("enter the price");
price=s.nextInt();
System.out.println("the color of any
transportaion
vechile");
color=s.next();}
void display()
{ System.out.println("The money is:"
+price);
System.out.println("the color
is:"+color);}}
class vehicles extends transportation
{String license;
int vehicle_no;
float distance;
float time;
Scanner s=new Scanner(System.in);
void read()
{ super.read();
System.out.println("The license of
the vechile");
license= s.next();
System.out.println("The vehicle_no of
an vechile");
vehicle_no=s.nextInt();
System.out.println("distance travelled");
distance=s.nextFloat();
System.out.println("time taken");
time=s.nextFloat();}
void display()
{ super.display();
System.out.println("The license
is:"+license);
System.out.println("The vechile_no
is:"+vehicle_no);
System.out.println("The
distance:"+distance);
System.out.println("The time
taken:"+time);}
public void calckilo()
{   float speed= distance/(float)time;
System.out.println("the
speed"+speed);}}
class car extends vehicles
{ String company;
float km;
int purchased_date;
int distance;
int time;
Scanner s=new Scanner(System.in);
void read()
{ super.read();
System.out.println("company name of
the car");
company=s.next();
System.out.println("The purchased_date
of the car");
purchased_date=s.nextInt();
System.out.println("The kilometer of an
vehicle
per liter");
km=s.nextFloat();
System.out.println("distance travelled");
distance=s.nextInt();
System.out.println("time taken");
time=s.nextInt();}
void display()
{ super.display();
System.out.println("The company name
of the car    is:"+company);
System.out.println("The purchased_date
of the car is:"+purchased_date);
System.out.println("The km of an
vehicle per liter is:"+km);
System.out.println("The
distance:"+distance);
System.out.println("The time
taken:"+time); }
public void calckilo() {int speed=
distance*time;
System.out.println("the
speed"+speed);}}
class Multipleinheritances
{public static void main(String[] args)
{ transportation t= new transportation
("sushu",1232);
car c = new car();
c.read();
c.display();
c.calckilo();
c.display(); }}
```

Figure 4 Shows the sample program of pair programmer

The Table 1 symbolizes the raw data. The information have been gathered from the result of the pair programmers

Table 1 shows the Pair Programming- Data set

Factors	P1	P2	P3	P4	P5	P6
File Size	4	3	2	3	2	4
Variable declaration	4	3	3	5	3	3
Errors	3	4	2	4	3	4
No of times debugging	4	3	2	3	3	2
Coupling Factor	0.942	0.944	0.939	0.955	0.948	0.952

Where

- File Size is the Size of program file.
- Variable declaration is the Number of variables declared in the program.
- Errors is the number of errors occurred during the Execution.
- Debugging is the Number of times the program debugged.
- $* \ Coupling \ Factor \ is \ computed$
- $1 - \frac{1}{di+2*ci+do+2*co+gd+2*gc}$

Figure 5 represents the outcome of pair programming for the sample dataset shown in Table 1

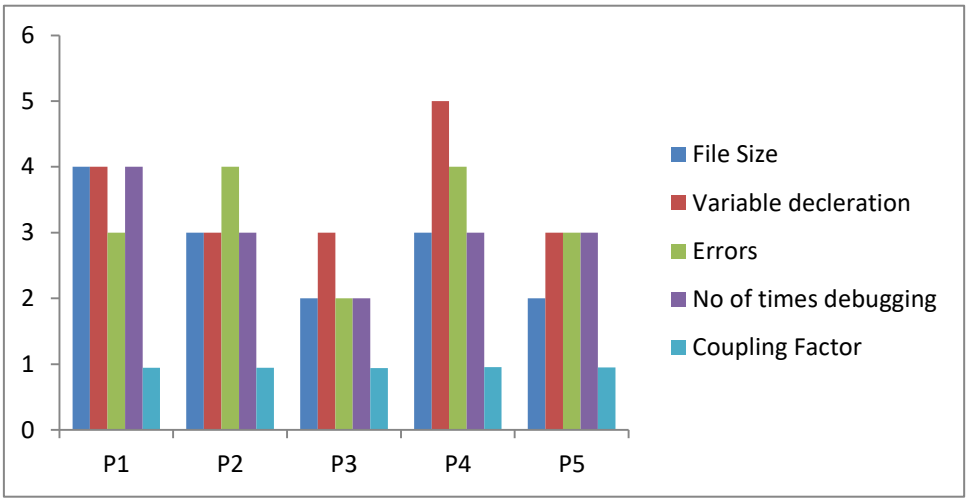

Figure 5 Shows the Outcome of Pair Programmers

4.2. TDD

For TDD Programming five programs which include the concepts like Multiple Inheritance, Sum of 'n' Numbers, transpose of matrix, a package sorting with selection and insertion sort, reversing a number and displaying the employee details are given. The following program is the primary program of a given set. It is developed by the TDD programmers' in which they have a chosen Transportation as a super class, Vehicle is a sub class of super class and it extended other two classes [car and bike]. Where each method is much dependent on one another, if interdependency is high coupling will be high and then the quality will be less.

```
import Vehicle.Driving;
import Vehicle.Owner;
import java.util.Scanner;
import java.util.*;
class Transporation
{ int price;
 String color;
 static Scanner s=new
Scanner(System.in);
void read()
{ System.out.println("The lisence of
vehicle");
 price=s.nextInt();
 System.out.println("The color of
vehicle");
 color=s.next();
}void display()
{ System.out.println("Lisence is:
"+price);
   System.out.println("Color is:"+color);
 }}
class Vehicles extends Transporation
implements
 Car,bike
```

```
{ int vehicleType;
   String fuel;
 void read()
   { try{
   super.read();
 System.out.println("Enter the
vehicleType");
 vehicleType=s.nextInt();
System.out.println("The type of fuel");
         fuel=s.next();}
      catch(Exception e)
      { }}
    public void disp()
    {super.display();
     System.out.println("The
vehicleType:"
       +vehicleType);
 System.out.println("Fuel:"+fuel); }
 Public void car()
  {  String car_no;
   System.out.println("The car_no");
car_no=s.next();
   System.out.println("car_no
is:"+criver_came);}
```

```
public void bike()

{   String bike_no;
   System.out.println("The bike_no");
   bike_noowner_name=s.next();
   System.out.println("bike_no
is:"+bike_no);
  }}
 class Interface
  { public static void main(String[]
args)
   {  vechile c =new vechile();
     c.read();
     c.display();
     c.car();
     c.bike();  }}

package Vehicle;
public interface car
{  public void car();}
package Vehicle;
public interface bike
{  public void bike(); }
```

Figure 6 Shows the sample program of TDD programmer

The Table 2 symbolizes the raw data. The information has been gathered from the result of the TDD programmers.

Table 2 Shows the TDD Programming- Data set

Factors	P1	P2	P3	P4	P5	P6
File Size	4	3	2	3	2	4
Variable declaration	4	3	3	4	2	3
Errors	8	6	4	5	5	4
No of times debugging	5	3	4	4	3	5
Coupling Factor	0.982	0.958	0.957	0.95	0.961	0.96

Where

- File Size- Size of program file.
- Variable declaration- Number of variables declared in the program.
- Errors- Errors occurred during the Execution.
- Debugging – Number of times the program debugged.
- $* \ Coupling\ Factor = 1 - \frac{1}{di+2*ci+do+2*co+gd+2*gc}$

Figure 7 represents the outcome of TDD programming for the sample dataset shown in Table 2

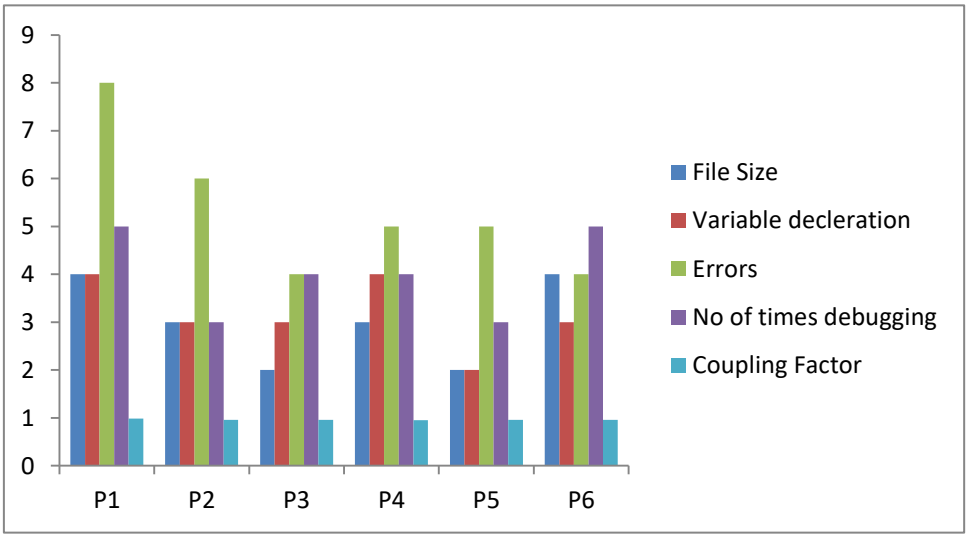

Figure 7 Shows the Outcome of TDD Programmers

5. CONCLUSIONS

To ensure the software quality, this paper mainly focused on the utility of Pair Programming and TDD. The analysis is performed with the programming experts on the basis of Development Time, Errors, Number of times debugged and coupling factor. The TDD Programmers took additional time, Number of Errors are high and coupling factor is also high than pair programmers. As the Pair Programmers results in less errors and coupling factor is low, from the outcome of PP and TDD Programming we can conclude that pair programming increases the software quality.

The proposed methodology is based on the two agile methodologies. Further this can be implemented to controlled studies on a bigger scale industry.

REFERENCES

[1] Body George, Raleigh, Laurie Williams. An initial investigation of test driven development in industry. North Carolina State University. ACM, **03**, 2003.

[2] E. Michael Maximilien and Laurie Williams. Assessing Test-Driven Development at IBM. IEEE, **03**, 2003.

[3] Jari Vanhanen, SoberIT and Harri Korpi. Experiences of Using Pair Programming in an Agile Project. IEEE, **07**, 2007.

[4] Laurie Williams. Pair Programming. North Carolina State University. ACM SIGCSE Bulletin, **39**, December 2007

[5] Laurie Williams, Robert R. Kessler, Ward Cunningham, Cunningham & Cunningham, Ron Jeffries. Strengthening the Case for Pair Programming. IEEE Software, **17**, July/Aug 2000.

[6] Thirumalesh Bhat, Nachiappan Nagappan. Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. ACM, **06**, 2006.

[7] Yahya Rafique and Vojislav B. Mi_si_c. The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis. Springer, 2014.

[8] https://www.agilealliance.org/glossary/pairing/#q=~(filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'pair*20programming))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

[9] http://agiledata.org/essays/tdd.html

[10] https://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=14&ved=0ahUKEwiotI_2j7jWAhUVSo8KHfmiDwsQFghwMA0&url=http%3A%2F%2Fagiledata.org%2Fessays%2Ftdd.html&usg=AFQjCNFycFr9cebS3TcpO4GskLTc9OasVA

[11] https://www.tutorialspoint.com/software_testing_dictionary/test_driven_development.htm

[12] https://www.guru99.com/test-driven-development.html

[13] Sunny Joseph Kalayathankal, John T Abraham and Jos eph Varghese Kureethara, An Ordered Ideal Intuitionisti c Fuzzy Software Quality Model, International Journal of Mechanical Engineering and Technology 8(10), 2017, pp. 535–546.

[14] Anand Handa, Ganesh Wayal, Software Quality Enhancement Using Fuzzy Logic with Object Oriented Metri cs In Design. International Journal of Computer Engineering and Technology (IJCET), 3(1), 2012, pp. 169 – 179

[15] R. Murugavel , Information Systems Software Quality: An Overview, International Journal of Mechanical Engineering and Technology 8(9), 2017, pp. 205 – 225

[16] Dr. S. Ravichandran, Design and Development of Software Fault Prediction Model To Enhance The Software Quality Level. International Journal of Information Technology and Management Information Systems (IJITMIS), (1), 2007, pp. 1–6