Volume 5, Issue 11, November 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Study of Software Testing Techniques for Reliability

Avinash H. Hedaoo*
Dept. of Computer Science
Dr. Ambedkar College, Nagpur, India

Abha Khandelwal
Dept. of Computer Science,
Hislop College, Nagpur, India

Abstract- ***Software Reliability is an important attribute of software quality, together with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. We should have adequate knowledge to choose among testing techniques as the activity of testing is expensive and time-consuming. We will prefer to select a technique that will achieve the goal in an effective and efficient manner. Different systematic testing methods should be evaluated for reliability as very limited knowledge of testing techniques in terms of their effectiveness and efficiency is available. The aim of this works is to study systematic testing techniques for reliability of software. This study can be considered as a significant step towards investigating which testing technique to select in systematic testing that will improve reliability as compare to other techniques in the product.***

Keywords - Software Testing Techniques to improve reliability, Software Reliability, Software Risk, Defect Detection techniques, systematic testing techniques

I. INTRODUCTION

Software Reliability is an important attribute of software quality, together with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve, because the complexity of software tends to be high. While the complexity of software is inversely related to software reliability, it is directly related to other important factors in software quality, especially functionality, capability, etc. Emphasizing these features will tend to add more complexity to software. [(1)]. In the studies [2],[3],[4],[5][6],[7] and [8] comparison of statistical testing with systematic testing has been done. The researchers believed that statistical testing might be more effective than debug testing for detecting failures, this was the original motivation for these studies. But these studies could not found conclusive result. Applying systematic testing techniques is necessary because, systematic testing not only increase software reliability but also achieve many other goals. Therefore, systematic testing techniques undoubtedly improve the reliability of the software which is being tested and achieve other goals also. We should have adequate knowledge to choose among testing techniques as the activity of testing is expensive and time-consuming. We will prefer to select a technique that will achieve the goal in an effective and efficient manner. According to the researcher [9] different systematic testing methods should be evaluated for reliability as very limited knowledge of testing techniques in terms of their effectiveness and efficiency is available. The aim of this works is to study systematic testing techniques for reliability of software. This study can be considered as a significant step towards investigating which testing technique to select in systematic testing that will improve reliability as compare to other techniques in the product.

II. SCHEMA USED

Many have called for an infrastructure for experimental software engineering [10]. The characterization scheme used in this thesis is proposed by [10] to carry out our experiment. The scheme is divided into four major parts, namely the goals and hypotheses that motivate an experiment, the plan for conducting the experiment, the procedures used during the experiment, and finally the results.

III. GOALS, HYPOTHESES AND THEORIES

In this study GQM (Goal – Question – Metrics) approach has been used to state the goal of the experiment. The idea behind the GQM Paradigm is that measurement should be based on goals. Following is the goal of our experiment based on GQM method :

Analyse **three defect detection techniques (systematic techniques)** for the **purpose of study** with respect to their **capability of increasing reliability(confidence)** from the point of view of the **researcher** in the context of a **controlled experiment** using small C programs.

The objective of the this experiment is to compare code reading, functional testing and structural testing techniques to improve the reliability by measuring number of defects detected during the experiment and individual weights associated with each failure which is explained in more detail in section VI. The aim of this study is to answer the question : Which testing criteria is more apt and helpful in terms of increasing confidence by decreasing the level of risk in the artifacts under test? In other words we can state that which testing technique improves the reliability of the software by a greatest factor.

A. Hypothesis

Testable hypotheses derived from goal are as follows :

H_{01} : Defect-detection techniques do not differ in improving the reliability of the software

H_{11} : Defect-detection techniques differ in improving the reliability of the software

By answering the following question above hypotheses can be tested :

Q1. In what ways does reliability of the software influence by each independent variable?

Q2. In what ways does time taken by software testing techniques to improve software reliability influence by each independent variable?

Q 3. Which technique isolated the largest percentage of faults from each severity level ?

B. Theories and Related Works

In [8] the researchers state that, most previous studies of the effectiveness of testing methods used the probability of causing a failure, and thus finding a defect, as a measure of the effectiveness of a test series. This seems inappropriate when considering testing as a means for improving the software: what really matters is the reliability of the delivered software, hence the improvement that is obtained by applying the given testing method.

In [11] the researchers analyzed the effect of input profile selection on software testing using the concept of fault detectability profile. They observed that optimality of the input profile during testing depends on factors such as the planned testing effort and the fault detectability profile. They state that to achieve ultra-reliable software, selecting test input uniformly among different input domains is preferred. On the other hand, if testing effort is limited due to cost or schedule constraints, one should test only the highly used input domains. Use of operational profile is also needed for accurate determination of operational reliability.

In [4] the researchers suggest to choose testing method based on delivered reliability obtained after all test failures have been eliminated. In [5] the analysis has shown the following points:

- There are obvious cases in which debug (systematic) testing is superior (roughly, because its detection rates are greater than the failure probability). Similarly, operational testing (statistical testing) can be obviously superior (roughly, because detection rates in many sub domains are smaller than the failure probability, so debug tests there are wasted). These examples show that the theory corresponds with intuition in limiting cases.
- If limited resources are available, only debug methods that focus on the most important failure regions are appropriate.
- In case of ultra-high reliability in safety-critical systems. While debug testing may be a means of identifying failure sets that have a very small chance of being encountered, and thus improving reliability beyond what can be achieved with operational testing, this cannot be guaranteed.

In [11][4] and [5] the researchers studied test effectiveness using different methods; the increment in reliability that would be obtained. In [12] the researcher uses expected failure cost under operational distribution as a measure for software (un)reliability. In the work of [12][13] and[14] a failure of any element of the i^{th} sub domain is assumed to cost c_i. In their work they divided the input domain using some partition testing strategy and associated a cost c_i a priori with each sub domain. However, this is not a realistic approximation of reality. In general, for the sub domains induced by the testing strategies commonly studied and in use, the consequences of failure for two different elements of the same sub domain may be very different. Furthermore, most sub domain testing strategies involve sub domains that intersect. In [15] to assess reliability of the software and to build an automatic test case generation algorithm to test the software cost/consequence of failure used as the basis. The failures must be categorized according to their significance to assess the risk associated with a program. Because in practice, some failures are trivial, while others are more severe, and some may even be catastrophic. Therefore, in this research, we assign a cost with each failure. Our work used both concept i.e. we also associate a cost to a failure and then based on the reduction in risk in the program evaluate testing techniques.

IV. EXPERIMENTAL PLAN

The experiment plan is used to illustrate the experiment design of the experiment. The treatments and objects used in the experiment are also indicated in experiment plan. The data collection and measurement techniques used in the experiment also described in it. An effective experiment plan helps to achieve the specified goals of the experiment as clearly and efficiently as possible.

A. Experimental Design

Subjects applied three defect-detection techniques(first independent variable) to three programs (second independent variable) in different orders (third independent variable). The experiment requires three days, but all subjects see the same program on the same day to prevent cheating. Therefore the variable "day" is confounded with "program" and is not considered as a separate independent variable. The uncontrolled variable subject is the forth independent variable.

Our dependent variables focus on counts of failures and the time spent to apply the techniques.

The randomization consists of assigning subjects randomly to one of three groups. Randomization refers to the random assignment of subjects to different levels of the independent variables. Membership in a group decides the match between technique and program for each subject as well as the order of applying the techniques.

The Experimental design used in our research study was randomized, fractional factorial design [19,20,21] . A combination of programs and techniques between the groups assures that each subject uses each technique(subjects are

measured three times), that all combinations of program and technique occur, and that all possible orderings of applying the techniques are seen. The Table I shows the Experimental Design Summary as follows :

TABLE I: EXPERIMENTAL DESIGN SUMMARY

ProgramDay	Program 1 (Day 1)	Program 2 (Day 2)	Program 3(Day 3)
Group 1	Structural Testing (ST))	Code Reading (CR)	Functional Testing (FT)
Group 2	Code Reading (CR)	Functional Testing (FT)	Structural Testing (ST)
Group 3	Functional Testing (FT)	Structural Testing (ST)	Code Reading (CR)

The design determines a minimum set of null hypotheses regarding external and internal validity of the experiment. The hypotheses concerning external validity correspond directly to the testable hypotheses derived from the goals; the rest check for threats to internal validity [19]. The experimental design allows us to test four null hypotheses in each case; these hypotheses will incorporate hypothesis stated in Section A of III.

i: The different techniques have no effect on the values of the dependent variables.
ii: The object (program) and day have no effect on the dependent variables.
iii: The order in which subject apply the techniques has no effect on the dependent variables.
iv: The subjects have no effect on the dependent variables.

The primary null hypothesis for external validity states that the different techniques have no effect on the values of the dependent variables. Additionally, null hypotheses concerning internal validity issues help the experimenter quantify threats such as selection or learning effects. For example, a null hypothesis may state that the different objects or subjects used in the experiment have no effect on the values of the dependent variables. Therefore, for evaluating the results based on the hypothesis and the questions stated in Section A following null hypothesis can be tested :

$N_{1.1}$: Independent variables (technique, program, group, or subject) do not differ in improving the software reliability.
$N_{1.2}$: Independent variables (technique, program, group, or subject) do not differ in time taken to improve the reliability .
$N_{1.3}$: Techniques do not differ in isolating faults of various severity levels.

B. Defect Detection Techniques

The classes of defect-detection techniques considered in this class of experiments are inspections and other reading (verification) techniques, functional testing, and structural testing. Each is discussed next. We use code reading, functional testing and structural testing. These three testing techniques also used in the study of [17][18][19]. All three techniques are applied in a two stage process. Firstly, failures are observed, that is the subject looks for observable differences between the program and the specification. Secondly, faults are isolated, that is the subject attempts to identify the exact location of the cause of failure in the program code.

1) Code Reading: Code reading is performed using stepwise abstraction without using any computer[20]. Subjects receive a line-numbered printed program but do not see the specification. Subjects form their own specification of the program by reading the source code . They identify prime subroutines (consecutive lines of code) write a specification for the subroutines as formally as possible, group subroutines and their specifications together, and repeat the process until they have abstracted all the source code. Finally, fault isolation step begins on the basis of inconsistencies observed (failure observation) during comparison of official specification with the specification written by the subjects (similar to failures in the other techniques).

2) Functional Testing : In step 1, Subjects are initially provided with an executable version of the program code and a program specification but do not see the source code. Functional testing was based on the standard techniques of equivalence partitioning and boundary value analysis. They identify equivalence classes in the input data and construct test cases based on the standard techniques of equivalence partitioning and boundary value analysis. In step 2, the subjects execute their test cases on the computer. They are instructed not to generate additional test cases during step 2, but we can neither prevent nor measure this. After having executed all the test cases subjects finish the Step 2 by taking print out of their results. In step 3, the subjects observe failures that were revealed in their output using the specification; no automatic test oracle was used[21]. After observing and recording the failures, the subjects hand in a copy of their printed output and receive the printed source code in exchange, and begin step 4. In step 4, the subjects use the source code to isolate the faults that caused the observed failures. No special technique is specified for the fault-isolation activity. Finally, subjects hand in a list of observed failures and isolated faults.

3) Structural Testing : Structural testing is also known as "clear box" or "white box" testing. In step 1, subjects are given only printed source code they do not receive the specification. In this technique subjects are asked to construct test cases to achieve 100% coverage of all branches, multiple conditions, loops, and relational operators(100% coverage is usually unrealistic). In step 2, the subjects execute their test cases using an executable version of the program and view output. Thereafter the subjects hand in a copy of their printed output, receive a copy of the specification in exchange, and begin step 3. In step 3, the subjects use the specification to observe failures in their output. In step 4, the subjects isolate the faults that caused the observed failures. No special technique is specified for the fault isolation activity. Finally, subjects hand in a list of observed failures and isolated faults.

C. Programs (Objects)

We used some self-coded small C programs, as training session was a learning process. Those programs were simple enough to understand and they were seeded with almost all types of faults. The program used in this study for experiment were approximately 300 lines long (excluding blank lines and comments). The three programs used in the experiment are as follows :

i. Library Management System (lms.c) : It is a simple library management system for college. It supports functions such Add book, Issue book, Search Book and Delete book etc. It also maintains data about books, students records that are required during various library operations.

ii. Personal Diary Management (pdm.c) : This program provides functionalities for managing personal information such Add record function for adding personal information, View Record function for getting information date wise or time wise and Delete Record function for deleting the record date wise or time wise etc.

iii. Telephone Bill Management (tbm.c) : The program manages records of the customers and generate their telephone bills. It provides functionalities like adding record, modifying record, listing record, searching record, paying telephone bills and deleting record etc.

The programs written in C, were nearly equal in length and use similar control constructs with which the subjects were familiar. In the Table III the cyclomatic complexity(the number of binary decision points plus one) measure for each program is given. Information about size data for the programs is shown in Table II.

TABLE II: SIZE AND OTHER RELEVANT INFORMATION FOR PROGRAMS

	lms.c	**pdm.c**	**tbm.c**
Total lines	315	276	270
Blank lines	12	5	9
Lines with comments	1	0	0
Non blanks non comment lines	295	267	256
Preprocessor directive	7	4	5

TABLE III: CYCLOMATIC COMPLEXITY INDEX

lms.c	**pdm.c**	**tbm.c**
33	31	33

1) Faults and Fault Classification : In this experiment we have seeded faults in each program which were supplied with Kamsties and Lott package. The programs may contain faults apart from seeded faults. All faults cause observable failures; no fault conceals another. The faults are chosen so that the programs fail only on some inputs, where a failure might be a total failure (no output at all), a serious problem (incorrect output), or a minor problem (misspelled word in the output). In this work the faults are classified using two-faceted fault classification scheme from [16] experiment. Facet one (type) captures the absence of needed code or the presence of incorrect code (omission, commission), and facet two (class) classified faults into six classes (initialization, computation, control, interface, data, cosmetic).

There were 8 faults in total in the lms program, 8 in pdm program, and 10 in the tdm program. Table IV classifies the faults in the programs used in this study according to the fault-classification scheme as discussed above.

TABLE IV: COUNT AND CLASSIFICATION OF FAULTS ACCORDING TO THE CLASSIFICATION

		lms	**pdm**	**tdm**	**Total**
Facet 1: Type	Omission	1	2	4	7
	Commission	7	6	6	19
Facet 2 : Class	Initialization	3	1	3	7
	Control	2	3	2	7
	Computation	0	0	1	1
	Interface	1	0	0	1
	Data	1	2	2	5
	Cosmetic	1	2	2	5
Total		8	8	10	26

2) Rules for counting failures and faults : There have been various efforts to determine a precise counting scheme for "defects" in software [16]. A failure is "revealed" if program output reveals behavior that deviates from the specification, and "observed" if the subject records the deviate behavior. In code reading, an inconsistency (analog to a failure in the other treatments) is "revealed" if the subject captures the deviate behavior in his/her own specification, and "observed" if the subject records the inconsistency between the specifications. Multiple failures (inconsistencies) caused by the same fault are only counted once. For all techniques, a fault is "isolated" if the subject describes the problem in the source code with sufficient precision. We distinguish between faults isolated by using the technique (i.e., after a failure was revealed and observed) and faults isolated by chance (i.e., no failure was revealed or observed), as summarized in Table V.

TABLE V: DIFFERENT DEFECT DETECTION AND ISOLATION CASES

Failure revealed?	Failure observed?	Fault isolated?	Interpretation
No	No	No	Defect-detection technique failed
No	No	Yes	Fault was isolated by chance
No	Yes	No	Meaningless
No	Yes	Yes	Code reading: constructed poor abstractions Functional/structural test: meaningless
Yes	No	No	Poor evaluation of abstractions/output
Yes	No	Yes	Fault was isolated by chance
Yes	Yes	No	Poor fault isolation
Yes	Yes	Yes	Defect-detection technique succeeded

D. Subjects

The subjects were the student of M.Sc. (Computer Science). They were well aware with C programming. Prior to the experiments the students were given lectures on each of the fault finding techniques during a 4 days (3 hours each day) workshop on Software Testing Techniques. Detailed instructions help train the subjects in applying each of the three defect-detection techniques. The subjects knew the basics of software testing, as they learnt the related subject (SE) during their academic session however they had not practical knowledge. The 18 subjects were selected for the experiment on the basis of participation, not a random one (accidental sample). Subjects were free to withdraw from the experiment at any point of time.

E. Data Analysis Procedures

In this study Parametric statistical procedures (ANOVA) were conducted using SPSS for Windows to test hypotheses. Analysis of variance (ANOVA) is a general method of drawing conclusions regarding differences in population means when two or more comparison groups are involved. Parametric techniques such as ANOVA are commonly used when randomization has been performed. Based on the randomized approach for matching techniques, programs and subjects, we used a parametric statistics method "ANOVA" to test the null hypotheses.

Like [18] we also included intermediate results to make analysis transparent. In all the tables in the subsection B of VII, SS refers to Sum of Squares and df refers to Degree of Freedom. In case of between groups column, SS refers to the treatment sum of squares and df refers to treatment degrees of freedom; whereas, in case of within group column, SS refers to the residual sum of squares, and df refers to the residual degrees of freedom. The computed F value is checked to see the p-value(significance level). We rejected the null hypothesis if we attained a probability value below the generally accepted cut off of 0.05. This refers to a 5% probability of mistakenly rejecting the null hypothesis[22].

V. EXPERIMENT PROCEDURES

This section explained the procedures used in conducting the experiment. In this section we have described precisely how the experiment will be performed.

The students were divided into 3 groups. Each group represents a set of students who performed the experiment individually on the same program at the same time applying the same technique. The live experiments were organized in 4-hours sessions over three days. All subjects are told that they will need a minimum of about four hours to complete each exercise. The results of the experiment (output data in the raw form) was recorded on data forms. An interactive session was taken to validate the subject's data after completion of the experiment. The interactive session was conducted to ensure process conformation i.e whether the subject applied the techniques as prescribed or not (process conformance), to investigate whether the subject supplied the data as per procedure (data validity) and check other problems to avoid misleading results.

A. Threats to validity

In the following way we have tried to eliminate threats to validity :

First we consider the problem of unknown factors that may influence the results without our knowledge, called threats to internal validity [22].

- To minimize maturation effects (a change in behavior over time) each subject tested each program using exactly one technique. However, the subject's comprehension of the C language could improve, during the experiment.
- Different types of programs and faults lead to instrumentation effect which can be measured to some extent.
- Selection effects was addressed by spreading the effects by random application of all techniques and matching groups and subjects randomly.
- Demotivation may also play a part as subjects become bored with three days of testing; however we strived to minimize it by asking subject to apply a new set of technique to a new program each day and by working for only 4 hours a day .
- Threats to external validity are factors which prevent the generalization of the results to actual software engineering practice. Subjects, programs and faults do not truly represent actual software engineering practice. These threads can only be addressed by repeated studies using true representatives of these factors.

VI. METHODOLOGY FOR STUDYING SOFTWARE RELIABILITY

The main objective of this study is to study systematic software testing techniques according to their effectiveness to detect and reduce the risk. To analyze the ability of testing criteria related to increase in reliability we assign a weight to each defect. In case of system failure during execution we can approximately define the relative effect or loss to the system because of that defect by assigning a weight to each defect(the severity of the failure). The goal of this study is to analyze the ability of systematic software testing criteria in terms of increase in reliability. We assign a weight to each defect because by having detected number of failures alone we can not compare the systematic testing techniques for reliability. The severity of the failures depends upon the cost incurred by, or damages inflicted on, the user. For example, any failure occurs due to a wrong word because of typing error in the output is of minor consequence, whereas a failure that surfaces during program execution results in wrong numerical value might be considered as of high severity. Even here, differences in the magnitude of the numerical inconsistency might lead to different forms of failure. For example, a failure in the form of wrong numerical value might be of fixed cost or might depend on how far the incorrect value is from the correct one.

Following categories of defect severity were taken into consideration:

i. Severity 1: Defects of this severity cause catastrophic consequences for the system. A defect of this level can cause program execution to abort and can result in critical loss of data, critical loss of system availability, critical loss of security, critical loss of safety, etc.

ii. Severity 2: Defects of this level cause serious consequences for the system. Such types of defects usually cause a program not to perform properly or to produce unreliable results. We also do not find a workaround for this type of defects. For example, a function is severely broken and cannot be used.

iii. Severity 3: Defects of this level cause significant consequences for the system. There is usually a work around for such type of defects till they are fixed. For example, losing data from a serial device during heavy loads.

iv. Severity 4: Defects of this level cause small or insignificant consequences for the system. Such defects are easily fixed and a workaround is available for them. For example, misleading error messages.

v. Severity 5: Defects of this level cause no negative consequences for the system. Such defects normally produce no erroneous outputs. For example, displaying output in a font other than what the customer desired.

We assigned a weight/cost of 1 to 5 to each defect. Weight/cost 5 is assigned to a failure of severity one, while weight/cost 1 is assigned to a failure of severity five(The weights are only an approximation as they depend on our perspective of failure consequence). The weights assigned to each defect of all the three programs depend on severity of the defect and corresponding weight/cost assigned to them. List and description of faults and failures and their corresponding weights for lms, pdm and tdm program have been shown in Table VI, Table VII and Table VIII respectively. It can be observed from the table VI, VII and VIII, faults of the same type and class need not have to be of the same weights assigned i.e the same defect severity. E.g. in Table VI, fault number F2 and F3 have been assigned with different weights i.e 5 and 2 respectively though they are of same type and class i.e. commission, control. Same in the case of fault number F4, F7 of Table VII and F4, F8 and F6, F7 of Table VIII.

TABLE VI: LIST AND DESCRIPTION OF FAULTS AND FAILURES AND THEIR CORRESPONDING WEIGHTS FOR LMS PROGRAM

Fault No.	Line No.	Fault Class	Fault Description	Causes failure	Weight/ cost assigned
F1	52	Commission cosmetic	The Close Application having wrong option i.e 6 it should be 5	The application can not be terminated normally	2
F2	57	commission Control	Case statement having wrong case constant i.e. 11 it should be 1	The addbook() function for adding books is not accessible	5
F3	93	commission Control	Condition is not checked properly i.e. in if statement '=' instead of '=='	In addbook option, when we enter choice, unable to enter information of book category	2
F4	103	commission Data	In fwrite () function wrong value 0 instead of correct value 1	In add book option when book is added it is not saved in the file	4
F5	119	Commission initialization	The variable "another" is initialized with wrong value i.e 'n' it should be 'y'	Unable to delete book, delete book option is not working	3
F6	222	Omission initialization	The variable "another" is not initialized, it should be initialized with value 'y'	Issue book option is not working properly	3
F7	228	Commission initialization	variable "another" is initialized with wrong value i.e 'n' it should be 'y'	Unable to issue book, issue book option is not working properly	3
F8	319	Commission interface	return statement returning wrong value i.e. 0 it should return correct value 1	Not allowing to add book	3

The faults are seeded in such a manner that they will not be biased towards the defects of the particular weights. After assigning the weights to the failures, we conducted the experiment as discussed in Section V. After that we evaluated which faults are detected and isolated by each technique. Only those defects were taken into account, which were revealed, observed and isolated by the subjects. For increasing the reliability isolation of the fault is necessary. Because risk is reduced only when the root cause of the failure is eliminated. After that we will perform the sum of weights of all failures detected and isolated for each technique for all the three programs and for each subject. A technique with highest weighted sum will be the one which have increased more confidence in the program as compared to others. The percentage of fault of each severity in lms.c, pdm.c and tdm.c program is shown in figure 1. The total weight for each technique will be calculated as:

Weighted Sum for technique X = 5*(Number of Severity 1 defects) +
4*(Number of Severity 2 defects) +
3*(Number of Severity 3 defects) +
2*(Number of Severity 4 defects) +
Number of Severity 5 defects - (1)

TABLE VII: LIST AND DESCRIPTION OF FAULTS AND FAILURES AND THEIR CORRESPONDING WEIGHTS FOR PDM PROGRAM

Fault No.	Line No.	Fault Class	Fault Description	Causes failure	Weight/ cost assigned
F1	37	Commission cosmetic	The Exit having wrong option i.e 5 it should be 4	The application cannot be terminated normally	2
F2	40	Omission cosmetic	User instruction is omitted. The prompt for user should be there	There is not instruction for user to operate the application	1
F3	102	commission Initialisation	The variable choice initialized with wrong value i.e. 1 it should be 0	Unable to add the record	5
F4	137	commission Control	The variable ch is compared with wrong value i.e 'n' it should be 'y'	Not allowing to add second record	4
F5	176	Commission Data	In fread function wrong value 0 is there, it should be 1	Unable to see record for the day in View record option	3
F6	194	Commission Data	In fread function wrong value 0 is there, it should be 1	Unable to see record for the fix time in View record option	3
F7	242	Commission Control	In case statement there is wrong value i.e. 11, it should be 1	Not allowing to delete whole record in Delete Record option	2
F8	260	Omission Control	The break statement is missing	When selecting to delete record by whole record option it is not functioning properly	3

TABLE VIII: LIST AND DESCRIPTION OF FAULTS AND FAILURES AND THEIR CORRESPONDING WEIGHTS FOR TDM PROGRAM

Fault No.	Line No.	Fault Class	Fault Description	Causes failure	Weight/ cost assigned
F1	43	Commission initialisation	Variable password declared which is not used in the program	No effect on the program	1
F2	60	Commission cosmetic	The Exit having wrong option i.e X it should be E	The application cannot be terminated normally	2
F3	78	omission Control	The break statement is missing	After deleting the record application not terminated normally	2
F4	117	commission Data	In fwrite () function wrong value 0 instead of correct value 1	In add record option when record is added it is not added in the file	5
F5	143	omission cosmetic	In printf function s.phonenumber variable is not written	List of record not displayed properly	2
F6	157	Omission initialisation	The variable "i" is initialized with wrong value i.e 0 , it should be initialized with value '1'	When deleting the record which is not existed appropriate message is not displayed	2
F7	202	omission initialization	The variable flag is not initialized, it should be initialized with value 1	When searching the record which is not existed search function is not working properly	3
F8	257	Commission data	In fseek function wrong argument is there i.e. SEEK_END it should be SEEK_CUR	Modify record option is not working properly. It is not modifying the current record instead appending the record in the list	3
F9	289	Commission	In for statement , incorrect	Line is not displayed	1

		control	operator is there i.e '>' , it should be '='		
10	296	Commission computation	Incorrect operator is there i.e. '-' , it should be '+'	Proper balance is not display when payment option is selected.	4

After applying the above formula according to the severity and weight assigned to the isolated faults of Table IX, we obtained weighted sum for each testing technique. The weighted sum for each testing techniques applied to three programs by each subject is as shown in Table X

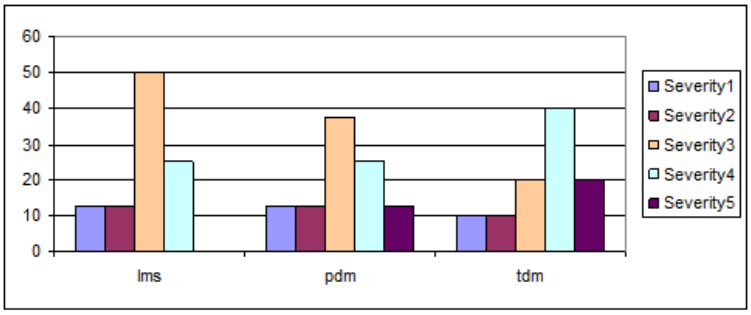

Figure 1: The percentage of fault of each severity

VII. RESULTS

This section examines the main findings of the experiment. The final aspects of any experiment are the raw data, the results of any inferential statistical analyses performed on the raw data, and interpretations of the statistical analyses [10].

A. Raw data

In this study, the raw data was collected from the experiment on the data forms. Table IX present the raw data of the experiment. In the Table IX, in each block first number represents the number of faults isolated and the second number represents the total time in minutes that was taken to isolate those faults respectively. Table X shows the weighted sum for testing techniques.

TABLE IX: TOTAL TIME TAKEN TO ISOLATE THE FAULTS

Sub-ject	Gro-up	Day 1 : lms			Day 2 : pdm			Day 3 : tdm		
		CR	FT	ST	CR	FT	ST	CR	FT	ST
1	1			3,225	4,230				7,225	
2	3		5,221				5,225	7,200		
3	2	3,230				6,230				6,208
4	1			4,220	4,235				6,236	
5	1			5,232	6,196				8,225	
6	2	3,235				5,238				5,235
7	2	4,230				3,192				5,213
8	3		5,180				6,236	6,232		
9	1			5,205	6,230				3,237	
10	3		6,209				4,232	8,237		
11	3		4,200				6,232	6,291		
12	2	5,232				6,231				7,236
13	1			3,215	6,232				5,186	
14	2	2,180				5,220				6,236
15	1			6,220	4,212				9,235	
16	2	5,215				3,190				6,236
17	3		6,235				3,160	5,145		
18	3		3,170				4,232	4,156		

In the table above, CR stands for Code Reading, FT stands for Functional Testing and ST stands for structural testing

B. Analysis and Interpretation

In this section results are analyzed and interpreted in an objective and critical way. All the results are illustrated clearly.

1) Analysis of effectiveness for improving reliability : For analyzing the effectiveness of software testing techniques for improving reliability, we can either accept or reject the null hypothesis $N_{1.1}$ of section A of IV . As shown in Table XI, null hypothesis is rejected with respect to the program (p=.014) only. Weighted sum for lms and pdm program is almost same whereas it is highest for tdm program suggests instrumentation effect and indicate the difference in the complexity of the programs. Weighted sum for functional testing is highest which is followed by structural testing, weighted sum is the least for code reading (relationship FT>ST>CR). Null hypothesis is accepted in case of technique, group and subject (as sig. level i.e $p > 0.05$).

2) Analysis of time taken to improve the reliability : For assessing the time taken to improve the reliability we can either accept or reject the null hypothesis $N_{1.2}$ of section A of IV. As show in Table XII we can reject null hypothesis with respect to group only, suggesting that there were significant maturation effect. Total time taken to improve the reliability increased on each day suggest presence of an instrumentation effect. This is because of programs differed in terms of complexity. Null hypothesis is accepted in case of technique, program and subjects.

TABLE X: WEIGHTED SUM FOR TESTING TECHNIQUES

Subject	Group	lms.c			pdm.c			tdm.c		
		CR	FT	ST	CR	FT	ST	CR	FT	ST
1	1			11	9				20	
2	3		18				16	18		
3	2	11				18				18
4	1			13	12				19	
5	1			17	17				22	
6	2	12				16				17
7	2	15				12				17
8	3		18				19	15		
9	1			16	16				12	
10	3		21				15	20		
11	3		13				18	18		
12	2	16				19				20
13	1			12	17				17	
14	2	9				16				18
15	1			20	12				23	
16	2	17				10				19
17	3		20				12	14		
18	3		12				14	13		

TABLE XI: ANALYSIS OF VARIANCE OF WEIGHTED SUM

Indepe-ndent Variable	Mean Weighted Sum			Between Groups		Within Groups		F	Sig. level
				SS	df	SS	df		
Technique	CR = 14.50	FT = 17.00	ST = 16.22	58.926	2	555.611	51	2.704	.077
Program (day)	LMS = 15.06	PDM =14.89	TDM = 17.78	94.70	2	519.833	51	4.646	.014
Group	G1 = 15.83	G2 =15.56	G3 = 16.33	5.593	2	608.944	51	.234	.792
Subject	13.33, 17.33, 15.67, 14.67, 18.67, 15.00, 14.67, 17.33, 14.67, 18.67, 16.33, 18.33, 15.33, 14.33, 18.33, 15.33, 15.33, 13.00, 15.90			165.87	17	448.667	36	.783	.700

TABLE XII: ANALYSIS OF VARIANCE OF TIME TAKEN TO IMPROVE THE RELIABILITY

Indepe-ndent Variable	Mean time taken to improve the reliability			Between Groups		Within Groups		F	Sig. level
				SS	df	SS	df		
Technique	CR =212.08	FT = 214.44	ST = 222.25	1019.176	2	28044.694	51	.927	.402
Program (day)	LMS= 214.25	PDM=219.61	TDM=214.91	307.343	2	28756.528	51	.273	.763
Group	G1 = 222.17	G2 = 221.44	G3 = 205.17	3326.926	2	25736.944	2	3.296	.045
Subject	226.67, 215.33, 222.67, 230.33, 218.67, 235.67, 211.67, 216.00, 226.00, 207.67, 233.00, 211.00, 212.00, 222.33, 213.67, 180.00, 186.00			10741.704	17	18322.167	36	1.242	.284

3) Analysis of ability of testing techniques for faults isolated of each severity level : For analyzing the ability of software testing techniques in terms of faults isolated of each severity level, we can either accept or reject the null hypothesis $N_{1.3}$ of section A of IV. We can reject null hypothesis with respect severity 2, severity 3 and severity 5, as shown in Table XIII. In the case of faults of severity 2 , functional and structural testers found to be most effective whereas code readers found to be least effective. Functional testing was most effective followed by structural testing which was followed by code reading at isolating faults of severity 3. But in the case of severity 5, code reading was most effective followed by functional testing which was followed by structural testing. All the three testing have isolated faults of severity 1 effectively. Null hypothesis is accepted in case of severity 4.

VIII. CONCLUSION

This empirical study can be considered as a significant step towards analyzing systematic software testing techniques for reliability. In this study after assessing three testing techniques, code reading, functional testing and structural testing for reliability we conclude that every testing technique contributes to reduction of risk in the software thus improving reliability of the software. It is found that functional testing is most effective in improving the reliability of the program. Code reading is most efficient in terms of time. The effect of the program was significant in case of the effectiveness for improving software reliability. Technique, group and subject had no significant effect on effectiveness in improving reliability. The effect of the group was found to be significant in case of the time taken to improve the reliability. Whereas technique, program, subject had no significant effect on efficiency in terms of time taken to improve the reliability.

The observed differences in effectiveness by severity level among the techniques suggest that a combination of the techniques might surpass the performance of any single technique.

To demonstrate that results are reliable and generalisable replication of this type of empirical study is necessary. The long-term goal of such work is to move software engineering from a craft towards an engineering discipline.

Therefore, it is required to further study software testing techniques but the study should be carried out in such a way that the results could be realistic and implementable. In this work, we aim to study software testing techniques for reliability. The problem addressed in this study is another step towards building a knowledge base which can help testing community to choose an effective and efficient testing technique to improve software reliability.

TABLE XIII: ANALYSIS OF VARIANCE OF PERCENTAGE OF ISOLATED FAULTS FROM EACH SEVERITY LEVEL

Fault Class	**Mean percentage of faults isolated of different severity levels**			**Between Groups**		**Within Groups**		**F**	**Sig. level**
	CR	**FT**	**ST**	**SS**	**df**	**SS**	**df**		
Severity 1	100.00	100.00	100.00	.000	2	.000	51		
Severity 2	83.33	100.00	100.00	3333.333	2	25000.00	51	3.4	.041
Severity 3	32.41	68.52	61.57	13218.028	2	55081.945	51	6.119	.004
Severity 4	47.22	26.39	41.67	4189.815	2	59201.389	51	1.805	.175
Severity 5	55.56	38.89	19.44	11759.259	2	97916.667	51	3.062	.055

REFERENCES

[1] Peter Liggesmeyer, Mario Trapp,"Trends in Embedded software engineering", Journal of IEEE software ,Vol. 26, Issue 3, 2009.

[2] Frankl P. and Weyuker E. A formal analysis of the fault-detecting ability of testing methods. Software Engineering, IEEE Transactions. 1993.

[3] Chen T. and Yu Y. On the expected number of failures detected by subdomain testing and random testing Software Engineering, IEEE Transactions. 1996

[4] Frankl P. Hamlet D. Littelewood B. and Strigini L.,Choosing a testing method to deliver reliability. 19th International conference on Software Engineering, ACM, 1997

[5] Frankl P. Hamlet D. Littelewood B. and Strigini L., Evaluating testing methods by delivered reliability[software]. Software Engineering, IEEE, 1998

[6] Frankl P. and Ikounenko O., further empirical studies of test effectiveness. In ACM SIGSOFT Software Engineering Notes, ACM 1998.

[7] Ntafos S., On random and partition testing. In ACM SIGSOFT Software Engineering Notes. Volume 23, ACM, 1998.

[8] Pizza M. and Strigini L. Comparing the effectiveness of testing methods in improving programs : the effect of variations in program quality. In Software Reliability Engineering, 1998, Procedings. The Ninth International Symposium, IEEE 1998.

[9] Umar Farooq, Evaluation Effectiveness of Software Testing Techniques with Emphasis on Enhancing Software Reliability, 2012

[10] Lott C. , Rombach H. Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques, Empirical Software Engineering, Spring 1997

[11] Li N. and Malaiya Y. On input profile selection for software testing. In Software Reliability Engineering, 1994, 5th International Symposium , IEEE, 1994.

[12] Gutjahr W. *Optimal test distributions for software failure cost estimation*. Software Engineering, IEEE 1995.

[13] Tsoukalas M., Duran J. and Ntafos S., *On some reliability estimation problems in random and partition testing*. Software Engineering, IEEE 1993.

[14] Ntafos S. *The cost of software failures* . IASTED Software Engineering Conference, 1997

[15] Weyuker E., *Using failure cost information for testing and reliability assessment*. ACM Transactions on Software Engineering and Methodology (TOSEM) 1996.

[16] Victor R. Basili and Richard W. Selby. *Comparing the effectiveness of software testing techniques*. IEEE Transactions on Software Engineering,13(12):1278–1296, December 1987.

[17] Juristo N. and Vegas S. *Functional testing, structural testing and code reading : what fault type do they each detect?* Empirical Methods and Studies in Software Engineering, 2003.

[18] Kamsties E. and Lott C. *An empirical evaluating of three defect-detection techniques*. Software Engineering ESEC'95, 1995.

[19] Roper M. Wood M and Miller J. *An empirical a valuation of defect detection techniques*.

[20] R.C. Linger, H.D. Mills, B.I. Witt, *Structured Programming: Theory and Practice*, Addison-Wesley, 1979.

[21] Elaine J. Weyuker. *On testing non-testable programs*. Computer Journal, 1982.

[22] Box G. Hunter J. and Hunter W. *Statistics for experimenters : design, innovation, and discovery*, volume 2. Wiley Online Library, 2005.

