Received: 8 November 2016 | Revised: 20 March 2017 | Accepted: 18 April 2017
DOI: 10.1002/smr.1883

WILEY Journal of Software: Evolution and Process

RESEARCH ARTICLE

A replicated empirical study to evaluate software testing methods

Sheikh Umar Farooq[1] | **S.M.K. Quadri**[2] | **Nesar Ahmad**[3]

[1] Department of Computer Sciences, University of Kashmir, Srinagar, India
[2] Department of Computer Sciences, Jamia Millia Islamia, New Delhi, India
[3] University Department of Statistics and Computer Appls, T.M. Bhagalpur University, Bhagalpur, India

Correspondence
Sheikh Umar Farooq, Department of Computer Sciences, University of Kashmir, Srinagar, India.
Email: suf.cs@uok.edu.in

Funding information
University Grants Commission (UGC), BSR Start-Up Grant. Grant Number: F.30-114/2015 (BSR)

Abstract
Many empirical studies have been performed to evaluate software testing methods in past decades. However, we are still not able to generalize the results as most studies are not complete and differ significantly from one another. To contribute to the existing knowledge base of software testing methods, we performed an empirical study to evaluate 3 testing methods: (1) code reading by stepwise abstraction, (2) functional testing using equivalence partitioning and boundary value analysis, and (3) structural testing using 100% branch, multiple-condition, loop, and relational-operator coverage using a well-defined and standard schema. A controlled experiment is performed with 18 subjects who applied the 3 defect detection techniques to 3 C programs in a fractional factorial experimental design to observe failures and isolate faults. The experimental results show that (1) the techniques are equally effective in terms of observing failures and finding faults, (2) the effectiveness of the techniques depends on the nature of the program, (3) all the testing techniques are equally efficient in case of failure observation, (4) the techniques are different in their efficiency in terms of fault isolation where code reading performed better than that of structural and functional testing, and (5) with respect to the fault types, all the techniques were equally effective in observing failures and isolating faults except in case of cosmetic faults where functional testing performed better than the other 2 techniques The effectiveness and efficiency of testing techniques were significantly influenced by the type of program. The results presented in this paper contribute to an empirical knowledge base of testing methods and may be helpful for the software engineers to decide the appropriate techniques in improving the software testing process.

KEYWORDS
code reading, comparison of testing methods, empirical study, functional testing, replication, structural testing

1 | INTRODUCTION

Most of the systems nowadays are becoming highly software intensive. Testing such systems requires us to competently decide which testing technique(s) to use. All defect detection techniques proposed so far can reveal failures, but we do not have adequate practical knowledge about them. Most of the information related to the techniques available is focused on how to apply the techniques but not on the applicability conditions of the techniques—practical information, suitability, effectiveness, efficiency, fault types they target, cost, etc. Furthermore, the benefits and drawbacks of each of the techniques are still quite unknown or at best unclear. For software engineering to move from a craft towards an engineering discipline, software developers need empirical evidence to help them decide what defect detection technique to apply under various conditions.[1,2] Empirical studies are crucial to software testing research to compare and improve software testing techniques and practices.[3,4] Several experiments have evaluated and compared various defect detection techniques. However, most of the existing experimental research on testing techniques does not provide a concrete basis for making any strong conclusions regarding the applicability conditions of different testing techniques as the results are generally inconclusive and conflicting. The experiments lack a formal foundation, and studies differ a lot in parameters they have taken into consideration. To avoid bias in comparison of the methods, the experiments should be conducted and reported under a common framework so that they can be compared in a way that can help us to generalize the results. Recent surveys on comparisons of various software testing techniques also conclude that further empirical research in software testing is needed, and that much more replication has to be conducted before general results can be stated.[5,6] Replications are crucial in empirical software engineering and help us with validating results of previously conducted experiments.[7-9] More replications of all types are required to increase the confidence of the results obtained from single experiment, as single experiment may not be able to answer all the questions under investigation. Replications guarantee to help us to derive stronger conclusions about the phenomenon under investigation. They are also used to transfer experimental knowledge, to train people, and to expand the base of experimental evidence.[10,11]

To contribute to the knowledge base of testing, we performed an exact-dependent* replication of an experiment that evaluates 3 software testing methods: *code reading*, *functional testing*, and *structural testing*. Dependent replications must come before independent replications (where researchers deliberately vary one or more major aspects of the conditions of an experiment) because, while either verifying or contradicting the original results, more insight can always be gained.[12] The aim of our replication is to gain confidence in results of previous studies by testing whether a given result is reproducible or not and thereby contributing to overall knowledge base on testing techniques. Our experiment aims to evaluate these techniques for effectiveness, efficiency, and fault types, which is discussed in Section 3.2 in detail.† The foundations of this experiment go back to the work of Hetzel[13] and Myers.[14] Most significant study in this sphere was conducted by Basili and Selby[15] who studied the effectiveness and efficiency of different code evaluation techniques. The work of Basili and Selby[15] was first replicated by Kamsties and Lott[16] who assumed the same working hypotheses as was in the experiment conducted by Basili and Selby,[15] but the experiment differed in many aspects like programming language, hypotheses, etc. In addition, the fault isolation‡ phase was added in the experiment conducted by Kamsties and Lott.[16] The work was again replicated by Roper et al[17] who used same guidelines, hypothesis, and laboratory package used by Kamsties and Lott.[16] Further, the experiment was replicated by Juristo and Vegas,[18] who stressed on fault types and did not consider efficiency of the testing techniques. The experiment was further replicated by Juristo et al[19] who stressed on effectiveness and other allied factors, which could affect the effectiveness of testing techniques. Multiple replications of an experiment increase the confidence in its results as it reduces variability in experimental results; it also increases the significance and confidence levels with which a researcher can draw conclusions and a practitioner can use them in realistic environment. Our effort is to add to the empirical evidence that will aid us in understanding the effectiveness and efficiency of testing techniques and its dependence on various factors. This study may help an industry practitioner to augment and validate his/her source of information about which approaches to testing should be applied in practice by focusing on the particular quantifications and comparisons provided by the results. It may also be useful to a researcher/experimenter to improve theories of software testing techniques as their theories on testing techniques are based on information gathered from a variety of sources. In addition, this study may also be helpful to a novice researcher/experimenter by serving as a tutorial, which shows how testing techniques can be empirically evaluated.

The rest of the paper is organized as follows: Section 2 presents the schema used for the experiment, Section 3 describes goals, hypotheses, and the related work, and Section 4 presents the experiment plan, which includes experimental design, testing techniques used, and programs used, described in Sections 4.1, 4.2 and 4.3, respectively. Section 5 describes the experiment procedures. Section 6 presents the results with data interpretation in Section 6.2, result summary in Section 6.3, and its comparison to previous work in Section 6.4, and finally, Section 7 presents conclusion and future work.

2 | SCHEMA USED

Over the years, many researchers have highlighted the need to perform experiments under a common framework, or using common/similar standards and parameters. In effect, we used a characterization schema proposed by Lott and Rombach[20] to perform our experiment. The schema is presented in Appendix A. Many of the guidelines proposed by Jedlitschka et al[21] and Carver[22] were also taken into consideration while replicating the experiment. Performing experiment using this schema will permit the meaningful comparison of results from similar experiments and establishes a context for cross experiment analysis of those results. This schema is also intended to reduce the effort required to develop and conduct further replications of experiments, which evaluate code reading, functional, and structural testing.

3 | GOALS, HYPOTHESES AND THEORIES

A statement of goals determines what an experiment should accomplish and thereby assists in designing and conducting the experiment.[23] We used Goal-Question-Metrics (GQM) approach to state the goals of our experiment. Accordingly, we define our main goal of the experiment as follows:

> Analyze code reading, functional testing, and structural testing for detecting software defects for the purpose of comparison with respect to their effectiveness and efficiency from the point of view of a researcher in context of a controlled experiment.

*A replication in which the procedures of an experiment are followed as closely as possible to original experiment and where researchers attempt to keep all the conditions of the experiment the same or very similar. Exact replications is sometimes referred as *strict* or *literal* or *close replication* by many researchers.

†Technique effectiveness was measured as the percentage of defects located by the technique over the total seeded faults and Technique efficiency was measured as the rate at which defects were found.

‡Detection refers to the observation of difference in the program's observed behavior and expected behavior. Isolation means to reveal root cause of the difference in the behavior i.e. *fault*.

3.1 | Goals

Goal 1. Effectiveness at revealing failures

The goal as per GQM is stated as follows: Analyze code reading, functional testing, and structural testing for the purpose of comparison with respect to their effectiveness at revealing failures/inconsistencies from the point of view of a researcher in context of a controlled experiment using small C programs.

Goal 2. Effectiveness at isolating faults

The goal as per GQM is stated as follows: Analyze code reading, functional testing, and structural testing for the purpose of comparison with respect to their effectiveness at isolating faults from the point of view of a researcher in context of a controlled experiment using small C programs.

Goal 3. Efficiency at revealing failures

The goal as per GQM is stated as follows: Analyze code reading, functional testing, and structural testing for the purpose of comparison with respect to their efficiency at revealing failures from the point of view of a researcher in context of a controlled experiment using small C programs.

Goal 4. Efficiency at isolating faults

The goal as per GQM is stated as follows: Analyze code reading, functional testing, and structural testing for the purpose of comparison with respect to their efficiency at isolating faults from the point of view of a researcher in context of a controlled experiment using small C programs.

3.2 | Hypotheses

Statements about the expected results that can be tested using an experiment are called testable hypotheses. To support testing such statements using inferential statistical methods, these statements are eventually formulated as null hypothesis, and the original statement is called the alternative hypothesis.[24] Our main goal is to study the effectiveness and efficiency of defect detection techniques. Accordingly, the 4 main hypotheses are

Testable hypotheses derived from goal 1 are as follows:

H_{01}: Subjects using defect detection techniques reveal and record the same percentage of total possible failures and as such do not differ in their failure observation effectiveness.[§]

H_{11}: Subjects using the defect detection techniques reveal and record a different percentage of total possible failures and as such differ in their failure observation effectiveness.

Testable hypotheses derived from goal 2 are as follows:

H_{02}: Subjects using defect detection techniques isolate the same percentage of faults after failures are observed by applying one of the defect detection techniques and as such do not differ in their fault isolation effectiveness.[¶]

H_{12}: Subjects using defect detection techniques isolate a different percentage of faults after failures are observed by applying one of the defect detection techniques and as such differ in their fault isolation effectiveness.

Testable hypotheses derived from goal 3 are as follows:

H_{03}: Subjects using defect detection techniques reveal and record a same percentage of total possible failures per hour and as such do not differ in their failure observation efficiency.

H_{13}: Subjects using defect detection techniques reveal and record a different percentage of total possible failures per hour and as such differ in their failure observation efficiency.

Testable hypotheses derived from goal 4 are as follows:

H_{04}: Subjects isolate a same number of faults per hour by applying one of the defect detection techniques and as such do not differ in the fault isolation efficiency.

H_{14}: Subjects isolate a different number of faults per hour by applying one of the defect detection techniques and as such do not differ in the fault isolation efficiency.

These hypotheses can be tested by answering the following questions:

Q1. What influence does each independent variable have on effectiveness of failure observation and fault isolation?

Q2. What influence does each independent variable have on the time to observe failures, time to isolate faults, and the total time?

[§]Only failures that were both revealed by the subject's detection efforts and were recorded by the subject are counted to compute this percentage.

[¶]An important requirement for counting isolated faults was that a failure corresponding to the isolated fault had been revealed. Without this requirement, the fault could have been isolated purely by chance, not based on the use of a defect detection technique.

Q3. What influence does each independent variable have on the efficiency of failure observation and fault isolation?

In addition to this, we will also extend our analysis to failure observation and fault isolation for each type and class of faults. This can be investigated by answering the following question:

Q4. Which technique leads to the observation of largest percentage of failures and isolation of largest percentage of faults from each type and class?

The metrics, which helped us to answer the above questions, which in turn helped us to verify our hypotheses, are reported in Section 4.1.

3.3 | Theories and related work

Seven studies are significant, which compares code reading, functional, and structural testing methods.[5,6] The description of each study is given in Table 1.[#] The table gives information about the number and type of the programs used in each experiment along with the number of faults in each program. Moreover, it also gives details about the type of techniques and number of subjects in each experiment. The summarized results obtained in each experiment are shown in the last column of Table 1. The effectiveness (average percentage of the defects found) and efficiency (average defect detected per hour) statistics for code reading, functional testing, and functional testing in each experiment are shown in Table 2 and Table 3, respectively. The hyphen (-) in some cells denotes that the corresponding parameter was not evaluated or measured in the experiment. If we have to conclude from the results obtained from these experiments, we will not be able to generalize all the results as they differ significantly from one another in most of the cases. The aim of this replication study is to further investigate these conclusions and to contribute to the body of empirical evidence that is evolving in this area.

4 | EXPERIMENTAL PLAN

The experiment plan specifies and describes the design of the experiment. In addition, it also specifies the treatments, objects, data collection, and measurement techniques used in the experiment. An effective experiment plan is very important to ensure that right type of data and a sufficient sample size are available to meet the specified objectives of the experiment as clearly and efficiently as possible.

4.1 | Experimental design

The experimental design consists of 4 independent variables (*defect detection technique*, *program*, *order*, and *subject*). All the subjects applied the techniques in different orders to the different programs. However, the same program was used on each day to avoid cheating. Our dependent variables mainly focused on failures detected and faults isolated as well as the time spent to detect the failures and to isolate the faults. The main dependent variables used in the experiment include:

1. *Number of failures detected.*
2. *Number of faults isolated.*
3. *Time to detect failures.*
4. *Time to isolate faults.*

Based on the dependent variables, following metrics were derived, which helped us to answer the questions, which in turn helped us to verify our hypotheses:

1. *Percentage of faults detected.*
2. *Percentage of faults isolated.*
3. *Time to detect faults.*
4. *Time to isolate faults.*
5. *Total time to detect and isolate.*
6. *Number of failures found/hour.*
7. *Number of faults isolated/hour.*
8. *Percentage of faults detected/type.*
9. *Percentage of faults isolated/type.*

[#]CR, FT, and ST stands for code reading, functional testing, and structural testing, respectively.

TABLE 1 Summary of existing studies

Author, Year	Programs	No. of Subjects	Techniques	Main Results	
		Fault Count		Aspect	Result
Hetzel 1976[13]	3 Programs coded in PL/I with 64 164 and 170 LOC	39 9, 15, and 25	Disciplined reading, selective testing, and specification testing	Effectiveness (detection)	CR < FT = ST
Myers 1978[14]	Single program coded in PL/I with 63 LOC	59 15	Walk-through/inspection, functional testing, and structural testing	Effectiveness (detection) Efficiency (detection)	CR = FT = ST ST < FT < CR
Basili and Selby 1987[15]	4 Programs coded in Simpl-T or Fortran with 169, 145, 147, and 365 LOC	74	Boundary value analysis, statement coverage, and stepwise abstraction	Effectiveness (detection)	1. CR > FT > ST (experienced subjects). 2. CR = FT > ST and CR = FT = ST (Case 1 and 2; inexperienced subjects). 3. Effectiveness also depends on software type.
		34		Efficiency (detection)	1. CR = FT = ST (experienced subjects). 2. ST < FT and CR (inexperienced subjects).
				Fault type	1. CR and FT > ST (omission and for initialization faults). 2. FT and ST > CR (interface faults). 3. FT > CR and ST (control faults). 4. CR > FT and ST (computation faults).
Kamsties and Lott 1995[16]	3 Programs coded C with 260, 279 and 282 LOC	50 (27 in replication 1 and 23 in replication 2)	Boundary value analysis, branch, multiple condition, loops and relational operators coverage and stepwise abstraction.	Effectiveness (detection)	Effectiveness depends on the program, not the technique.
				Effectiveness (isolation)	Effectiveness depends on the program and subject, not on the technique
		36 (11,14,11 in replication 1 and 6,9,7 in replication 2)		Efficiency (detection)	1. Functional took less time than structural technique. 2. Fault detection time also depends on the subject.
				Efficiency (isolation)	Efficiency depends on the program and subject, not on the technique
				Fault type	For both detected and isolated: No significant difference was observed between techniques
Roper et al. 1997[17]	3 Programs coded C with 260, 279 and 282 LOC	47	Boundary value analysis, branch coverage, and stepwise abstraction	Effectiveness	1. Effectiveness depends on the program/technique combination 2. Effectiveness depends on nature of faults
		8,9,8		Efficiency	Functional and structural performed better than code reading
Juristo and Vegas 2003[18]	4 Programs coded in C with 400 LOC	196 9*4	Equivalence partitioning, branch coverage, and stepwise abstraction	Effectiveness (detected and observable	1. Effectiveness depends on program, technique, and fault. 2. FT and ST > CR (for the defect type). While FT = ST. 3. The program version influences on the number of subjects that detect a defect.
Juristo et al. 2012[19]	3 Programs coded in C with 209, 172, and 146 LOC.	8 Replications with 42, 39, 29, 35, 31, 31, 172, and 76 subjects. 7 Each	Equivalence partitioning, branch coverage, and stepwise abstraction	Effectiveness	1. FT = ST > CR. 2. The effectiveness of code reading by stepwise abstraction varies significantly from program to program. 3. FT using equivalence partitioning and ST using branch testing are independent of the fault type.

TABLE 2 Average percentage of defects detected in existing experiments

		Effectiveness		
		Code Reading	Functional Testing	Structural Testing
Hetzel, 1976[13]		37.3	47.7	46.7
Myers, 1978[14]		38	30	36
Basili and Selby, 1987[15]		54	54.6	41.2
Kamsties and Lott, 1995[16]	Replication 1	43.5	47.5	47.4
	Replication 2	52.8	60.7	52.8
Roper et al, 1997[17]		32.1	55.2	57.5
Juristo and Vegas, 2003[18]	Replication 1	19.98	37.7	35.5
	Replication 2	-	75.8	71.4
Juristo et al, 2012[19]		47.23	79.26	78.43

TABLE 3 Average defect detection rate (per hour) in existing experiments

		Efficiency		
		Code Reading	Functional Testing	Structural Testing
Hetzel, 1976[13]		-	-	-
Myers, 1978[14]		0.8	1.62	2.07
Basili and Selby, 1987[15]		Depends on program	Depends on program	Depends on program
Kamsties and Lott, 1995[16]	Replication 1	2.11	4.69	2.92
	Replication 2	1.52	3.07	1.92
Roper et al, 1997[17]		1.06	2.47	2.20
Juristo and Vegas, 2003[18]	Replication 1	-	-	-
	Replication 2	-	-	-
Juristo et al, 2012[19]		-	-	-

The experimental design applied to our study was randomized-fractional-factorial design. It involved 3 factors (defect detection technique, program, and the order in which these techniques were applied). The randomization consists of match of techniques, programs, order of applying the techniques and assigning subjects randomly to 1 of the 3 groups. A place in a group decides the match between technique and program for each subject as well as the order of applying the techniques. Each group represents a set of people who performed the experiment individually on the same program at the same time applying the same technique. Repeated measurements taken as subjects were observed multiple times (within-subject design) as every subject applied each technique in the experiment.

Our experiment like many other previous experiments measured the performance of every subject on each combination of the 3 programs and the defect detection techniques. However, it was considered that once a subject applies one detection technique to a program, there would be a learning effect, as the subject will acquire some knowledge about the program and some of its faults. Thus, it was not reasonable to let the subject apply another detection technique to the same program. This restricts the experimental design to the combinations in which each subject applied one technique to one program as shown in Table 4.

The hypotheses concerning external validity correspond directly to the testable hypotheses derived from the goals; the rest check for threats to internal validity.[20] The experimental design allows us to test 4 null hypotheses in each case; these hypotheses will incorporate the hypothesis stated in Section 3.2.

D$_1$: *The technique has no effect on the results (ie, the techniques do not differ in their effectiveness and efficiency).*

D$_2$: *The program and the day has no effect on the results; ie, no selection effects.*

D$_3$: *The order in which subjects apply the techniques has no effect on the results; ie, no learning effects.*

D$_4$: *The subjects have no effect on the results; ie, all subjects perform similarly (no selection effects).*

The primary null hypothesis for external validity in our case states that the different techniques have no effect on the results. Additionally, null hypotheses concerning internal validity issues help the experimenter quantify threats such as selection or learning effects. As an example, a null hypothesis may state that the different programs (or the subjects) have no effect on the results.[20] So, depending on the hypothesis and questions stated in Section 3.2, we will analyze the results by testing the following null hypothesis:

N$_{1.1}$: *None of the independent variables (technique, program, group, or subject) affect the percentage of total possible failures observed.*

N$_{1.2}$: *None of the independent variables (technique, program, group, or subject) affect the percentage of total faults isolated.*

N$_{1.3}$: *None of the independent variables (technique, program, group, or subject) affect time taken to reveal and observe failures.*

N$_{1.4}$: *None of the independent variables (technique, program, group, or subject) affect time taken to isolate faults.*

N$_{1.5}$: *None of the independent variables (technique, program, group, or subject) affect the total time, which includes both failure detection and fault isolation time.*

TABLE 4 Experimental design summary

Program (day)	Program 1 (Day 1)	Program 2 (Day 2)	Program 3 (Day 3)
Group 1	Code reading (CR)	Functional testing (FT)	Structural testing (ST)
Group 2	Functional testing (FT)	Structural testing (ST)	Code reading (CR)
Group 3	Structural testing (ST)	Code reading (CR)	Functional testing (FT)

$N_{1.6}$: *None of the independent variables (technique, program, group, or subject) affect the failure observation rate.*

$N_{1.7}$: *None of the independent variables (technique, program, group, or subject) affect the fault isolation rate.*

$N_{1.8}$: *Techniques are equally effective at observing failures caused by the various types and classes of faults.*

$N_{1.9}$: *Techniques are equally effective at isolating faults of the various types and classes of faults.*

4.2 | Defect detection techniques

We used the same defect detection techniques as used in the experiment performed by Kamsties and Lott[16]: *code reading*, *functional testing*, and *structural testing*. Different experiments in the past have used different criteria for each testing method. In our case, *code reading* was applied using stepwise abstraction, *functional testing* used equivalence class partitioning, and boundary value analysis and *structural testing* used 100% branch, multiple-condition, loop, and relational-operator coverage. For example, 100% coverage of a multiple condition using a single logical and operator means that all 4 combinations of true and false must be tested, and 100% coverage of a loop means that it must be executed zero, one, and many times. For *structural testing*, our subjects have not used any tool, like Generic Coverage Tool (GCT), which was used by Basili and Selby[15] and Kamsties and Lott[16] in their experiments to measure coverage. This affects the time it takes the subjects to generate the test cases (but not the quality of the task performance, as the programs are simple enough for subjects to test without a testing tool).[18] All techniques are applied in a 2-stage process: *failure observation* (observable differences between the program and the official specification) and *fault isolation* (identifying exact location of the cause of a failure in a program code).

4.3 | Programs

No programs from Kamsties and Lott package[16] were used in the training session, which was conducted to give subjects an introduction of the experiment. Instead, we used some self-coded trivial programs, as training session was more a learning process rather than a pilot study. Those programs were simple enough to understand and they were seeded with almost all types of faults types, which were seeded in the programs used in the actual experiment. The 3 programs we used in the experiment were same as used by other works[16-18] and Juristo et al[19] in their live experiments. They are as follows:

1. *Cmdline*: Program that reads the input at the command line and evaluates the input and fills a data structure with results of evaluation and outputs a summary.
2. *Nametbl*: Program that implements an abstract data structure of a symbol table, as well as its operations. The operations include inserting a symbol, setting its attributes, searching for a symbol by name, and printing out the contents.
3. *Ntree*: Program that implements the data structure of an n-ary tree. The operations include creating a tree, inserting a node, searching and querying the tree, and printing out the contents of the tree.

All the programs were written in C language with which the subjects were familiar. Table 5 gives quantitative data related to the size of the programs. In addition, to compare relative complexity of the programs, cyclomatic complexity index measure (the number of binary decision points plus one) is also shown for each program. Code readers may have faced difficulties as most of the programs were without comments.

TABLE 5 Size and other relevant information for programs

		Cmdline.c	Cmdline.h	Nametbl.c	Nametbl.h	Ntree.c	Ntree.h
Total Lines		245	34	251	31	235	25
Blank Lines		26	5	40	8	38	7
Lines with comments		0	0	4	0	4	0
Nonblank/noncomment lines		219	29	207	23	193	18
Preprocessor directives		3	17	6	3	5	4
Cyclomatic complexity index	Minimum	1		1		1	
	Mean	7		2		3	
	Maximum	28		8		8	

4.3.1 | Faults and fault classification

The faults used in our experiment were supplied with package developed by Kamsties and Lott.[16] Although faults present in the program were seeded, we still cannot guarantee that programs did not contain any other faults. As we wanted to evaluate the effectiveness of the testing methods with respect to fault types also, a fault classification was needed to classify the faults. We classified the faults using the 2-faceted fault-classification scheme used by Basili and Selby[15] experiment. Facet 1 (type) captures the absence of needed code or the presence of incorrect code (omission, commission). Facet 2 (class) partitions software faults into the 6 classes:

1. *Initialization.*
2. *Control.*
3. *Computation.*
4. *Interface.*
5. *Data.*
6. *Cosmetic.*

There were 8 faults in total in *ntree* program, 9 in *cmdline* program, and 8 in *nametbl* program. Table 6 classifies the faults in the programs used in the experiment as per the above classification.

4.3.2 | Failure counting scheme

We used same fault counting scheme as used in Kamsties and Lott experiment[16] as shown in Table 7, ie, we only count those faults for which a failure is revealed, observed, and then the fault is successfully isolated. Fault isolated due to chance was not taken into consideration. In addition, failures caused by the same fault were counted only once.

4.4 | Subjects

Eighteen students pursuing their Masters in the Department of Computer Sciences, University of Kashmir participated in the experiment. The subjects of the experiment were selected on the basis of participation, not randomly. The subjects were the final semester students and were

TABLE 6 Count and classification of faults as per adopted classification

	Cmdline	Nametbl	Ntree	Total
Omission	2	5	4	11
Commission	7	3	4	14
Initialization	2	0	0	2
Control	2	3	3	8
Computation	0	1	0	1
Interface	3	1	1	5
Data	1	2	2	5
Cosmetic	1	1	2	4
Total	**9**	**8**	**8**	**25**

TABLE 7 Different defect detection and isolation cases

Failure Revealed?	Failure Observed?	Fault Isolated?	Explanation
N	N	N	Defect-detection technique failed
N	N	Y	Fault was isolated by chance
N	Y	N	Meaningless
N	Y	Y	Code reading: constructed poor abstractions functional/structural test: meaningless
Y	N	N	Poor evaluation of abstractions/output
Y	N	Y	Fault was isolated by chance
Y	Y	N	Poor fault isolation
Y	Y	Y	Defect-detection technique succeeded

already familiar with software testing. The subjects also had a minimum of 2 years of experience in programming (including the C language). The subjects were equally experienced and had more or less same profile.

The low number of subjects was indeed a threat to validity; however, because of low number of students willing to participate in the experiment, we had to perform with aforementioned number of subjects only. Actually, 22 students registered for voluntary participation in the experiment; however, only 21 were present in training session, and later on, only 18 students turned up for the experiment session. Those who left were not asked to explain why they left the experiment.

All participants were aware of the fact that this experiment is for research purpose. Participation was totally based on the consent of the subjects as no subject was forced to join the experiment. Subjects were free to withdraw from the experiment at any point in time. They were assured that they would not be judged personally based on their performance.

4.5 | Data collection and validation procedures

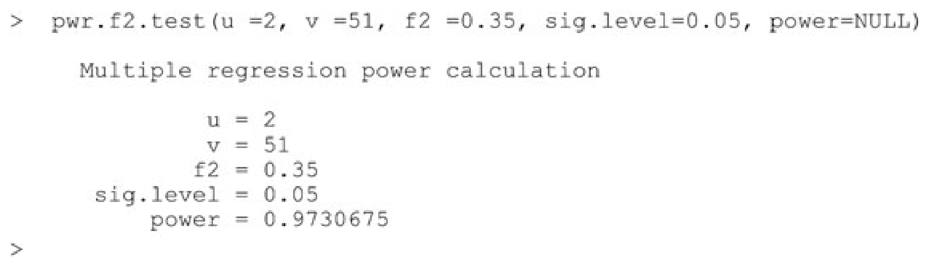
The output data (the result in raw form) were collected via data collection forms, which were used by Kamsties and Lott[16] in their experiment. However, they were customized to make them more suitable for the experiment. After the subjects completed the experiment, we validated their data by arranging a small interactive session with them. The aim was to check conformance to the process, validity of the data, and other issues that might lead to erroneous or misleading results. This also helped us in compiling raw data into the usable form.

4.6 | Data analysis procedures

We used *SPSS* statistical package for data analysis. Parametric statistics method analysis of variance (ANOVA) was used to test the null hypotheses because our experiment design involved the randomization of defect detection techniques, programs, and subjects, and all the analyses involved 3 or more groups (variables) of data. Like Kamsties and Lott,[16] we also included intermediate results to make analysis transparent. In all the tables in Section 6.2, *SS* refers to sum of squares and *df* refers to degree of freedom. However, 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. All these values are used to make a test against the F distribution to check whether the variation between the treatments exceeds that of the variation within the treatment. The F value is defined as the ratio of the variance between groups to the variance within groups. Each variance is calculated as the sum of squares divided by its degree of freedom. In short, the F value is computed as *SS*/*df* (between groups)/(*SS*/*df* (within groups). The computed value is used to check the *P* value (significance level). The significance level is an estimate of whether the difference is due to chance. An important issue is the choice of significance level, which specifies the probability of a result for being realistic. However, it does not specify that the results are highly important. Common practice dictates rejecting the null hypothesis when the significance level is less than or equal[25] to .05. We also rejected the null hypothesis if we attained a probability value below the generally accepted cut off value .05. This refers to 5% probability of mistakenly rejecting the null hypothesis also known as “Type I error.”

Moreover, we also addressed the concern of committing “Type II error,” and as such, we estimated the power of the test. We computed the power for the statistical test using pwr.f2.test in R package. However, to compute power, effect size is required first. For effect size computation, we used partial eta-squared (η_p^2) measure. Cohen suggested that d = 0.2 be considered a “small” effect size, 0.15 represents a “medium” effect size, and 0.35 a “large” effect size. We used the same classification in our experiment. On average, the effect size for all cases from Sections 6.2.1 to 6.2.7 is large. Using the large effect size value, we found that power of test is well above the accepted value of 0.90 as shown in Figure 1. This clearly indicates the lesser chances of committing Type II error and suggests that more chances of the null hypothesis not being rejected are because it is actually true (ie, there is no actual difference in the testing methods).

The power of test was also calculated for each case separately (Sections 6.2.1 to 6.2.7) using Minitab 17. The results for all the cases are well above the acceptable level of 0.90. The results are shown in Appendix B.



```
>  pwr.f2.test(u =2, v =51, f2 =0.35, sig.level=0.05, power=NULL)

     Multiple regression power calculation

              u = 2
              v = 51
             f2 = 0.35
      sig.level = 0.05
          power = 0.9730675

>
```

FIGURE 1 Power of test computation in R

5 | EXPERIMENT PROCEDURES

Experiment procedures are the details of implementation of the experiment design. Experimental procedures describe precisely how the experiment will be performed.

5.1 | Training activities

Before running the experiment, 3 training sessions of 2 hours each were performed with the subjects to give them an introduction to the experiment. As already mentioned, training session was more a learning process rather than a pilot study. We had a good discussion on various aspects of the experiment and also on the testing methods to be used. All subjects had a minimum of 2 years of experience in programming (including the C language). In addition, the subjects were also taught how to fill in the data collection forms. The training session gave them a good insight into the working and purpose of the experiment.

5.2 | Conducting the experiment

The live experiment was run in 3 parallel sessions in 3 different labs on 3 consecutive days under our close observation. All subjects were aware of 4-hour (240 minutes) time limitation for performing the task. The experiment was conducted at Department of Computer Science, University of Kashmir. All 18 subjects participated in the experiment until the very end.

5.3 | Threats to validity

We tried our best to eliminate maximum threats to validity. In spite of that, some threats to validity could not be excluded. The threats to validity are discussed as per the classification given by Wohlin et al.[26]

5.3.1 | Construct validity

To mitigate this threat to validity, we used the same established metrics for measurements as were used in the previous experiment by the researchers. The measurements were taken in an objective manner and the metrics correspond to the intended meanings of the measured variables in theoretical terms.

5.3.2 | Conclusion validity

1. We used subjects with the same experience and similar academic background. No subject had any previous experience with the controlled experiments. Thus, we ensured that the sample was homogenous.
2. Moreover, the statistical tests in the experiment were properly used without violating the assumptions like applying parametric test ANOVA after checking data for normality.
3. The low number of subjects could pose a threat to validity, which is very common in Software Engineering experiments; however, because of a low number of students willing to participate in the experiment, we had to perform with aforementioned number of subjects only. The power of test was computed to ensure that the experiment has a high statistical power.

5.3.3 | Internal validity

1. Maturation effects (like learning) have been eliminated, as each subject applied a different technique to a different program on each day. However, the subject's comprehension of C language may have got improved during the experiment.
2. In addition to that, there may have been some fatigue effects causing subjects to lose motivation. There was no way to measure it; however, we tried to minimize the effect by using a different set of techniques and programs each day and by limiting the experiment to only 4 hours a day.
3. All the subjects were equally experienced and had a similar academic background. The subjects were very well familiar with the C language in which programs were written. Moreover, random match of subject, group, and the order of applying the techniques minimized selection effects.
4. There were some instrumentation effects as the programs had different complexities and contained different types of faults; analysis of variance attempts to measure this effect.

5.3.4 | External validity

External threats to validity were more dominant in our case. Subjects, programs, and faults did not truly represent actual software engineering practice. However, as reported by Host et al,[27] software engineering students may be used instead of junior software professionals for experimentation purposes provided they have adequate knowledge about language, techniques to be used in the experiment. So subjects do not really pose a big

TABLE 8 Raw data for failure observation effectiveness and efficiency

		Day 1: Program–Nmdline			Day 2: Program–Nametbl			Day 3: Program–Ntree		
Subject	**Group**	**CR**	**FT**	**ST**	**CR**	**FT**	**ST**	**CR**	**FT**	**ST**
1	2		5,120				4,105	6,152		
2	2		4,94				3,152	4,128		
3	3			3,121	3,164				4,160	
4	2		3,81				6,164	6,142		
5	1	5,126				3,98				5,111
6	3			8,154	4,152				5,148	
7	3			5,143	5,149				6,152	
8	1	6,143				5,127				4,122
9	1	5,128				3,105				6,109
10	2		6,102				4,94	7,152		
11	3			6,72	3,82				5,127	
12	2		7,114				5,132	7,160		
13	2		5,117				5,142	5,129		
14	1	7,132				6,124				7,162
15	3			3,86	5,128				4,81	
16	1	2,112				4,97				7,167
17	3			5,92	3,97				5,161	
18	1	3,102				5,104				6,164

The pair of numbers represents the number of failures observed and the time (in minutes) that was taken by subjects to reveal and observe those failures.

threat to external validity. The threat to validity emerging from using same small sized programs with high fault density could not be mitigated because of the exact dependent nature of the experiment. Replicating the experiment in an independent dissimilar fashion using true representatives of these factors can minimize the external threat to validity.

5.4 | Giving feedback to subjects

A feedback session was held 2 months after the experiment. The feedback session was held late because of the unavailability of most of the subjects. We thanked all the subjects for their participation in the experiment. The subjects asked many questions and in return thanked us for the practical knowledge they gained through this experiment.

6 | RESULTS

The final outcome of any experiment is the raw data, the results of any inferential statistical analysis performed on the raw data, and interpretations of the statistical analysis.[20] Analysis of variance was used to test all the null hypotheses in Sections 6.2.1 to 6.2.7. We rejected the null hypothesis if we attained a probability value below the generally accepted cut off value .05. This refers to 5% probability of mistakenly rejecting the null hypothesis also known as "Type I error." The concern of committing "Type II error" was also addressed and as such.

Moreover, we also addressed the concern of committing "Type II error," and as such, we estimated the power of the test, which is well above the accepted value of 0.90. This clearly indicates the lesser chances of committing Type II error and suggests that more chances of the null hypothesis not being rejected is because it is actually true (ie, there is no actual difference in the testing methods).

6.1 | Raw data

Raw data (also known as source or atomic data) are unprocessed data collected from the sources. In our case, the raw data were collected from the experiment using data forms. Raw data require selective extraction, organization, and sometimes analysis and formatting for presentation before it can be used. Table 8 and Table 9 present the raw data of the experiment obtained after processing and formatting information obtained from the data forms used in the experiment.[‖] The pair of numbers in each block in Table 8 represents the number of failures observed by a subject using a technique applied on corresponding program and the time (in minutes) that was taken by subjects to reveal and observe those failures. The pair of numbers in Table 9 represents the number of faults isolated by a subject after using a particular technique on a corresponding program and the time taken by subjects to isolate those faults.

[‖]CR stands for code reading, FT stands for functional testing, and ST stands for structural testing

TABLE 9 Raw data for fault isolation effectiveness and efficiency

		Day 1: Program–Cmdline			Day 2: Program–Nametbl			Day 3: Program–Ntree		
Subject	**Group**	**CR**	**FT**	**ST**	**CR**	**FT**	**ST**	**CR**	**FT**	**ST**
1	2		2,62				3,32	5,27		
2	2		3,47				3,49	4,18		
3	3			1,26	3,28				3,63	
4	2		3,38				4,46	5,20		
5	1	5,24				3,38				3,32
6	3			5,39	3,35				5,81	
7	3			3,41	5,46				5,79	
8	1	6,21				4,82				3,45
9	1	5,22				3,78				4,38
10	2		4,56				3,44	6,28		
11	3			3,26	3,21				3,82	
12	2		6,67				5,65	7,22		
13	2		4,54				4,43	5,29		
14	1	7,18				5,73				6,47
15	3			3,23	4,34				3,47	
16	1	2,27				4,68				4,53
17	3			4,32	3,12				5,79	
18	1	3,17				4,76				6,42

The pair of numbers represents the number of faults isolated and the time (in minutes) taken by subjects to isolate those faults.

6.2 | Interpretation

Researchers should describe their results clearly, in a way that other researchers can compare them with their own results. Results need to be interpreted in an objective and critical way before assessing their implications and before drawing any conclusions.

6.2.1 | Evaluation of failure observation effectiveness

For evaluating the effectiveness of software testing methods for observing failures, we evaluate the null hypothesis $N_{1.1}$: *None of the independent variables (technique, program, group, or subject) affect the percentage of total possible failures observed.* We can reject null hypothesis with respect to the program only (instrumentation effects) as the significance level is below .05. As shown in Table 10, effectiveness on day 1 and day 2 is almost same; however, a 30% increase in effectiveness on day 3 for *ntree* program may be an indication of the presence of possible learning effect. We failed to reject null hypothesis in case of technique, group, and subject, which suggest that the techniques were equally effective with regard to failure revelation and observation.

6.2.2 | Evaluation for fault isolation effectiveness

For evaluating the effectiveness of the software testing methods for isolating faults, we evaluate the null hypothesis $N_{1.2}$: *None of the independent variables (technique, program, group, or subject) affect the percentage of total faults isolated.* We can reject null hypothesis again with respect to the program only (instrumentation effects). Increase in effectiveness in isolating faults on each day also suggests possible learning effect. However, as shown in Table 11, the increase factor is not uniform. We failed to reject null hypothesis in case of the technique and the group, which suggest that the techniques were equally effective in case of fault isolation.

TABLE 10 Analysis of variance of percentage of failures observed

Independent Variable	Mean Percentage of Total Failures Observed			Between Groups		Within Group			
				SS	df	SS	df	F	Sig Level
Technique	CR = 57.56	FT = 56.71	ST = 61.57	242.698	2	14392.25	51	0.43001	0.652
Program (day)	cm = 54.32	na = 52.77	nt = 68.75	2794.139	2	11840.81	51	6.01736	**0.004**
Group	G1 = 59.64	G2 = 61.57	G3 = 55.16	388.588	2	15564.34	51	0.63664	0.533
Subject	60.18, 43.98,40.27, 61.11, 51.85, 67.12, 64.35, 59.72, 56.01, 68.05, 47.2, 75.92, 60.18, 80.09, 48.61, 53.24, 51.85, 56.94			5622.607	17	9431.584	36	1.26242	0.270

TABLE 11 Analysis of variance of percentage of faults isolated

Independent Variable	Mean Percentage of Total Faults Isolated			Between Groups SS	Between Groups df	Within Group SS	Within Group df	F	Sig Level
Technique	CR = 54.08	FT = 46.21	ST = 45.06	868.698	2	12112.26	51	1.82887	0.170
Program(day)	cm = 42.59	na = 45.83	nt = 56.94	2039.609	2	10941.36	51	4.75352	**0.012**
Group	G1 = 51.31	G2 = 51.08	G3 = 42.97	810.828	2	12170.14	51	1.69892	0.193
Subject	40.74, 40.27, 28.70, 48.61, 43.51, 51.85, 52.77, 51.38, 47.68, 52.31, 36.11, 72.22, 52.31, 71.75, 40.27 40.74, 48.14, 52.77			6046.811	17	6934.156	36	1.84665	0.601

6.2.3 | Evaluation of time taken to observe failures

For evaluating the software testing methods for time taken to observe failures, we evaluate the null hypothesis $N_{1.3}$: *None of the independent variables (technique, program, group, or subject) affect the time taken to observe the failures*. We can reject null hypothesis with respect to the program as shown in Table 12. Increase in the time taken on each day to reveal and observe the failures points towards the instrumentation affects and indicates the difference in complexity of the programs. We failed to reject null hypothesis in case of the technique, the group, and the subject, which suggests that the techniques do not differ with respect to the time taken to reveal and observe the failures.

6.2.4 | Evaluation of time taken to isolate faults

For evaluating the software testing methods for time taken to isolate faults, we evaluate the null hypothesis $N_{1.4}$: *None of the independent variables (technique, program, group, or subject) affect the time taken to isolate the faults of the observed failures*. We can reject null hypothesis with respect to the technique only as shown in Table 13. The results suggest that code reading took much less time to isolate time followed by structural testing, functional testing took the longest time (CR < ST < FT), which is obvious because the code readers take long time to develop their own specifications and identify inconsistencies, after which they isolated faults rapidly, whereas the functional testers reveal and observe failures rapidly, but then they usually take more time to isolate faults because they had to comprehend the code for that purpose. We failed to reject null hypothesis in case of the program, the group, and the subject, which suggest that they had no effect on the time taken to isolate the faults.

6.2.5 | Evaluation of total time (detection time + isolation time)

With respect to the total time taken to detect failures and isolate faults, we evaluate the null hypothesis $N_{1.5}$: *None of the independent variables (technique, program, group, or subject) affect the total time, which includes failure observation time and fault detection time*. We can reject null hypothesis with respect to the program only as shown in Table 14 (instrumentation effects). The total time taken by subjects to detect and isolate faults increased on each day, which clearly suggests the presence of an instrumentation effect. The result supports the statement of subjects that programs significantly differed in terms of complexity. We failed to reject null hypothesis in case of the technique, the group, and the subject.

TABLE 12 Analysis of variance of failure-observation time

Independent Variable	Mean Failures Observation Time			Between Groups SS	Between Groups df	Within Group SS	Within Group df	F	Sig Level
Technique	CR = 132.11	FT = 117.33	ST = 127.33	2047.259	2	34857.78	51	1.4976	0.233
Program (day)	cm = 113.27	na = 123.11	nt = 140.38	6781.37	2	30123.67	51	5.7405	**0.005**
Group	G1 = 124.05	G2 = 126.66	G3 = 126.05	67.1481	2	36837.89	51	0.0464	0.954
Subject	125.66, 124.66, 148.33, 129.00, 111.66, 151.33, 148.00, 130.66, 114.00, 116.00, 93.66, 135.33, 129.33, 139.33, 98.333, 125.33, 116.66, 123.33			12855.7	17	24049.33	36	1.132	0.364

TABLE 13 Analysis of variance of fault-isolation time

Independent Variable	Mean Fault Isolation Time			Between Groups SS	Between Groups df	Within Group SS	Within Group df	F	Sig Level
Technique	CR = 24.94	FT = 65.00	ST = 40.16	14717.15	2	6877.44	51	54.5678	**0.000**
Program (day)	cm = 35.55	na = 48.33	nt = 46.22	1689.037	2	19905.56	51	2.16374	0.125
Group	G1 = 44.50	G2 = 41.50	G3 = 44.11	95.814	2	21498.78	51	0.11364	0.892
Subject	40.33, 38.00, 39.00, 34.66, 31.33, 51.66, 55.33, 49.33, 46.00, 42.66, 43.00, 51.33, 42.00, 46.00, 34.66, 49.33, 41.00, 45.00			2173.926	17	19420.67	36	0.23704	0.998

TABLE 14 Analysis of variance for total time

Independent Variable	Mean Total Time			Between Groups SS	Between Groups df	Within Group SS	Within Group df	F	Sig Level
Technique	CR = 157.61	FT = 182.33	ST = 167.50	5574.037	2	59900.78	51	2.3728	0.103
Program (day)	cm = 149.38	na = 171.44	nt = 186.61	12611.81	2	52863	51	6.0836	**0.004**
Group	G1 = 169.11	G2 = 168.16	G3 = 170.16	36.03704	2	65438.78	51	0.0140	0.986
Subject	166.00, 162.66, 187.33, 163.66, 143.00, 203.00, 203.33, 180.00, 160.00, 158.66, 136.66, 186.66, 171.33, 188.66, 133.00, 174.66, 157.66, 168.33			20818.81	17	44656	36	0.9872	0.492

6.2.6 | Evaluation of efficiency in observing failures

For evaluating the efficiency of the testing methods in case of failure observation, we evaluate the null hypothesis $N_{1.6}$: *None of the independent variables (technique, program, group, or subject) affect the rate of failure observation*. In this case, we have to accept null hypothesis for all the factors as shown in Table 15. The results in principle imply that none of the independent variables affect the mean failure observation rate. However, as the significance value for program is just above .05 level for failure observation efficiency (ie, .058), we can consider it as significant and assume that the program affects the failure observation rate. The results therefore suggest that we failed to reject null hypothesis in case of the technique, group, and the subject.

6.2.7 | Evaluation of efficiency in isolating faults

For evaluating the efficiency of the testing methods in case of fault isolation, we evaluate the null hypothesis $N_{1.7}$: *None of the independent variables (technique, program, group, or subject) affect the rate of fault isolation*. We can reject the null hypothesis with respect to the technique. Moreover, the Wilcoxon test (see Appendix C) revealed that there is a significant difference between code reading, functional, and structural testing. Code reading was the most efficient followed by structural testing technique. Functional testing was the least efficient. However, we failed to reject null hypothesis in case of the program, the group, and the subject as shown in Table 16.

The box plots for the fault isolation percentage (derived based on number of faults isolated), fault isolation time, and fault isolation efficiency are shown in the Figures 2–4. The box plots in Figure 2 reveal interesting features of fault isolation percentage data. First, the variability differs substantially from code reading to structural testing. Second, there are quite a few outliers in structural testing, and hence, the effect of structural testing does not appear to be normally distributed. Third, the code reading is able to isolate maximum faults followed by functional and structural testing.

The box plots in Figure 3 reveal that the effect of functional testing is highly variable than code reading and structural testing. It is also clear that the mean time taken to isolate faults is higher in case of functional testing followed by structural and code reading. However, code reading took the least amount of time to isolate faults, which were observed by it. Therefore, as code reading was able to isolate maximum faults in a less time, the fault isolation efficiency of code reading is higher than other 2 testing methods.

TABLE 15 Analysis of variance of mean failure observation rate

Independent Variable	Mean Failure Observation Rate			Between Groups SS	Between Groups df	Within Group SS	Within Group df	F	Sig Level
Technique	CR = 2.157	FT = 2.461	ST = 2.496	1.25106	2	22.5782	51	1.4129	0.252
Program (day)	cm = 2.631	na = 2.103	nt = 2.380	2.5122	2	21.3170	51	3.005	0.058
Group	G1 = 2.372	G2 = 2.452	G3 = 2.290	0.23578	2	23.59175	51	0.2568	0.774
Subject	2.38, 1.87, 1.36, 2.31, 2.30, 2.24, 2.15,2.28, 2.45, 2.94, 3.18, 2.86, 2.33, 2.89, 2.46, 2.02, 2.32, 2.28			9.014752	17	14.81458	36	1.2886	0.254

TABLE 16 Analysis of variance of mean fault isolation rate

Independent Variable	Mean Fault Isolation Rate			Between Groups SS	Between Groups df	Within Group SS	Within Group df	F	Sig Level
Technique	CR = 11.783	FT = 3.6419	ST = 5.6746	646.443	2	323.2218	51	32.9111	**0.000**
Program (day)	cm = 7.93	na = 5.460	nt = 7.701	67.2756	2	1080.041	51	1.58839	0.214
Group	G1 = 7.728	G2 = 7.508	G3 = 5.863	37.3891	2	1109.92	51	0.8589	0.429
Subject	6.22, 6.94, 3.86, 8.31, 7.62, 5.51, 4.90, 8.02, 7.41, 7.07, 5.89, 9.69, 6.79, 11.70, 6.23, 4.16, 8.76, 7.43			188.71	17	958.6041	36	0.41688	0.971

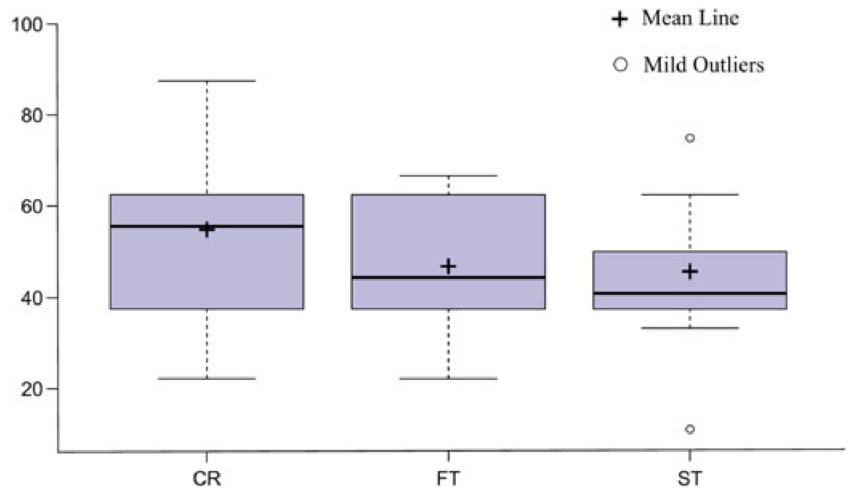


FIGURE 2 Fault isolation percentage

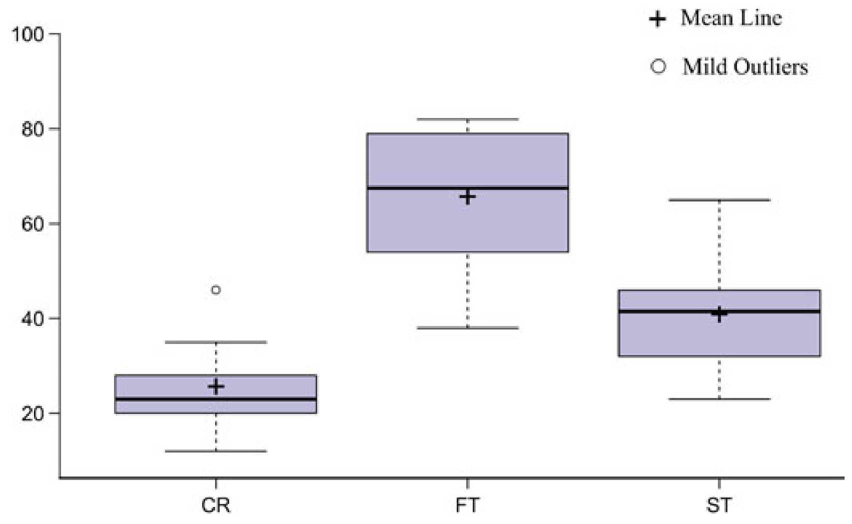


FIGURE 3 Fault isolation time

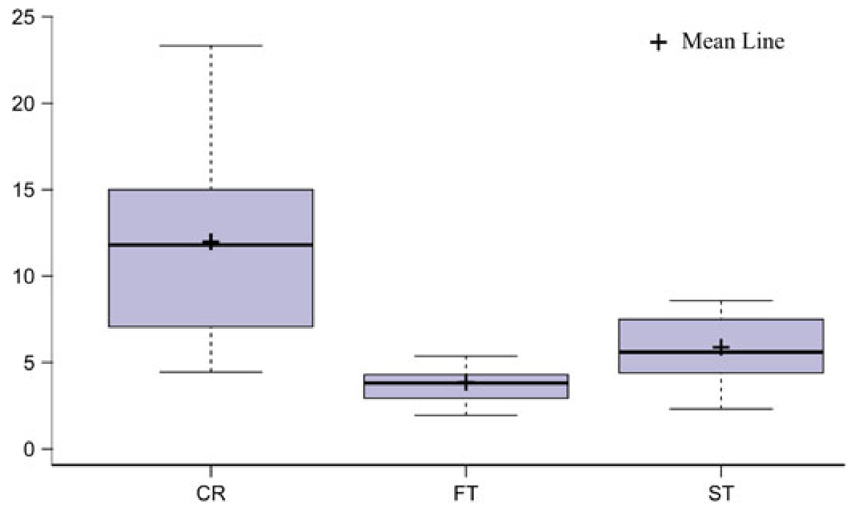


FIGURE 4 Fault isolation efficiency

Figure 4 reveals that the variance is greater in code reading than in functional and structural testing. However, structural testing is the second best efficient technique despite having less mean fault isolation percentage than functional as functional testing took far more time to isolate the observed faults. As such time has a clear influence on the fault isolation efficiency. As was the case with original experiment,[16] code reading in our experiment was also a 3-step process, whereas the functional and structural testing was a 4-step process. Moreover, structural testing requires knowledge of source code, so isolating faults becomes comparatively easy in comparison to functional testing where source code knowledge is not required at all.

6.2.8 | Evaluation of effectiveness of failures observed for each fault class

For evaluating the effectiveness of testing methods in terms of failure observation for each fault class and type, we evaluate the null hypothesis $N_{1.8}$: *Techniques are equally effective at observing failures caused by the various types and classes of faults*. We can reject the null hypothesis in case of cosmetic faults as shown in Table 17 where structural testing out performed code reading. Functional testing was the most effective in terms of observing cosmetic faults.

6.2.9 | Evaluation of effectiveness of faults isolated for each fault class

For evaluating the effectiveness of testing methods in terms of fault isolation for each fault class and type, we evaluate the null hypothesis $N_{1:9}$: *Techniques are equally effective at isolating faults of the various types and classes of faults*. We can reject the null hypothesis in case of cosmetic faults only as shown in Table 18. In this case, functional testing isolated maximum number of faults followed by structural testing, which was followed by code reading.

6.3 | Summary of the results

Our results suggest that the techniques do not differ significantly in case of failure observation effectiveness and fault isolation effectiveness, and as such, we accept the null hypothesis H_{01} and null hypothesis H_{02}, respectively. Our results suggest that the effectiveness of failure detection and fault isolation depends on the type of program and not on the technique.

The time taken to observe the failures also depends on the nature of the program only; however, the time taken to isolate faults depends on the technique; the results suggest that functional testing using boundary value analysis took the longest time while the code reading took least time to isolate the faults from observed failures (CR < ST < FT). The total time depends only on the program.

As the significance value for program is just above .05 level for failure observation efficiency (ie, .058), we can consider it as significant and assume that the program affects the failure observation rate. The efficiency of failure observation (mean failure observation rate) depends on the type of the program only and as such we accept the null hypothesis H_{03}. However, the efficiency of fault isolation (mean fault isolation rate) depends on the technique only; code reading performs better than functional and structural testing (CR > ST > FT), and as such, we reject the null hypothesis H_{04}. One justification for better performance of code reading in case of fault isolation is that the code reading is less sensitive to fault

TABLE 17 Analysis of variance of percentage of observed failures caused by faults from each fault class

	Mean Percentage of Observed Failures			Between Groups		Within Group			
Fault Class	CR	FT	ST	SS	df	SS	df	F	Sig Level
Omission	63.05	61.94	73.61	1492.593	2	35754.17	51	1.0645	0.35
Commission	58.59	61.24	61.50	93.2224	2	34358.78	51	0.0691	0.933
Initialization	22.22	16.66	25	648.148	2	67361.11	51	0.245361	0.783
Control	58.33	60.18	67.59	864.197	2	34922.84	51	0.63102	0.536
Computation	27.77	33.33	33.33	370.37	2	116111.1	51	0.08134	0.922
Interface	68.51	61.11	61.11	658.43	2	10482	51	0.160852	0.851
Data	69.44	50	52.77	3981.481	2	93055.56	51	1.0910	0.343
Cosmetic	41.66	88.88	77.77	21944.44	2	70138.89	51	7.97821	**0.000**

TABLE 18 Analysis of variance of percentage of isolated faults caused by faults from each fault class

	Mean Percentage of Isolated Faults			Between Groups		Within Group			
Fault Class	CR	FT	ST	SS	df	SS	df	F	Sig Level
Omission	59.16	56.38	56.38	92.59	2	33093	51	0.071348	0.931
Commission	55.35	45.96	44.51	1247.79	2	27748.09	51	1.146702	0.325
Initialization	22.22	5.55	11.11	2592.59	2	33333.33	51	1.983333	0.148
Control	54.62	48.14	52.77	401.234	2	26635.8	51	0.384125	0.683
Computation	22.22	33.33	33.33	1481.481	2	111111.1	51	0.34	0.713
Interface	68.51	51.85	33.33	11152.26	2	112654.3	51	2.5243	0.090
Data	63.88	38.88	36.11	8425.92	2	95833.33	51	2.2420	0.116
Cosmetic	41.66	88.88	75	21203.7	2	70277.78	51	7.6936	**0.001**

hiding than the other 2 techniques. The process of code reading almost condenses failure detection and fault isolation into one activity. Functional and structural testing process requires additional efforts after failure detection by delving into the source code to isolate a fault.

With respect to fault type and class of fault detected and isolated, all testing methods were equally effective except in case of cosmetic faults where functional testing was most effective and code reading was least effective (FT > ST > CR).

6.4 | Comparison with related work

Table 19 summarizes and compares the results of our experiment with the results of Kamsties and Lott[16] experiment. We agree with the results obtained by Kamsties and Lott in most of the cases. However, we also disagree or contradict results obtained by them in some cases. In comparing our results with the results of previous studies, we believe that besides independent variables, many other factors can contribute to the differences in results, like the nature and procedure of the experiment or any alteration made to the original experiment. Overall, we have following mixed conclusions.

1. We support the results of Kamsties and Lott experiment for effectiveness that the techniques did not differ significantly in terms of failure observation effectiveness (hypothesis H_{01}) and fault isolation effectiveness (hypothesis H_{02}).
2. We failed to observe a significant difference between techniques in case of effectiveness, which is very much in line with the results of the related work, discussed in Section 3.3 with an exception of Basili and Selby experiment[15] for experienced subjects case only and Hetzel.[13]

TABLE 19 Comparison of results with the actual experiment

Factor Studied	K&L (Kamsties and Lott)	Farooq et al	Confirm
Failure observation effectiveness	R1: CR = FT = ST(T) R2: CR = FT = ST	CR = FT = ST(T)	Confirmed
Fault isolation effectiveness	R1: CR = FT = ST(T) R2: CR = FT > ST	CR = FT = ST(T)	Confirmed with replication 1
Failure observation time	R1: CR > FT > ST(T) R2: CR > FT > ST(T)	CR = FT = ST(T)	Not Confirmed
Fault isolation time	R1: FT > ST > CR R2: FT > ST > CR	FT > ST > CR	Confirmed
Total time (detection + isolation)	R1: CR = FT = ST(T) R2: CR = FT > ST(T)	CR = FT = ST(T)	Confirmed with replication 1
Failure observation efficiency	R1: FT > ST > CR R2: FT > ST > CR(T)	CR = FT = ST(T)	Not Confirmed
Fault isolation efficiency	R1: FT > CR = ST(T) R2: FT > CR > ST	CR > ST > FT	Not Confirmed
Fault types observed			
Omission	R1: ST = FT > ST|R2: All	CR = FT = ST	Confirmed with R2
Commission	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Initialization	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Control	R1: FT = ST > CR|R2: All	CR = FT = ST	Confirmed with R2
Computation	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Interface	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Data	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Cosmetic	R1/R2: CR = FT = ST	FT > ST > CR	Not Confirmed
Fault types isolated			
Omission	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Commission	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Initialization	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Control	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Computation	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Interface	R1: CR > FT > ST|R2:All	CR = FT = ST	Confirmed with R2
Data	R1/R2: CR = FT = ST	CR = FT = ST	Confirmed
Cosmetic	R1: All: R2: FT > ST > CR	FT > ST > CR	Confirmed with R2

Symbols “R1” and “R2” stand for “replication 1” and “replication 2.”
Symbol “=” stands for “mean values did not differ significantly.”
Symbol “>” stands for “left value is greater than right value.”
Symbol “T” stands for “threats to internal validity were seen.”

3. We cannot support the Kamsties and Lott[16] experiment for the failure observation efficiency (hypothesis H_{03}) and fault isolation efficiency (hypothesis H_{04}). Our results differ in both the cases.
   a. In case of failure observation efficiency, Kamsties and Lott experiment like Roper et al[17] found significant differences in techniques; our results suggest that they all performed equally, which are in line with the results of Basili and Selby.[15]
   b. With respect to fault isolation efficiency, our results suggest that code reading performed better than structural testing, which outperformed functional testing. In case of Kamsties and Lott, functional testing performed better than the other 2 techniques.
4. In case of effectiveness of failures observed for each fault class, we confirm the results of Kamsties and Lott experiment in most of the cases except for the cosmetic fault type, where we found that functional testing was most effective followed by structural and code reading, which was least effective. Our results like Kamsties and Lott differ from the results of Basili and Selby.[15]
5. In case of effectiveness of faults isolated for each fault class, we support the results of Kamsties and Lott experiment for all the cases.
6. Our results are consistent with the results of previous studies that the type of a program influences the effectiveness and efficiency of techniques significantly.

Table 1 and Table 19 comparison will help in understanding our work with the other previous related studies.

7 | CONCLUSION AND FUTURE WORK

In this paper we compared the effectiveness and efficiency of 3 software testing methods—*code reading* by stepwise abstraction, *functional testing* using equivalence partitioning and boundary value analysis, and *structural testing* using 100% branch, multiple-condition, loop, and relational-operator coverage with a well-defined schema.

We conclude that all methods are equally effective in case of failure observation and fault isolation as we failed to observe a significant difference between techniques. These results are very much in line with the results of the previous studies conducted over the past 3 decades, which confirms that the failure observation and fault isolation effectiveness does not depend on the technique but on the type of a program. We believe that this view is now becoming a generalized view in the empirical software engineering community.

We also failed to observe a significant difference between the techniques with respect to failure observation efficiency. Our results in this case differed with the result of Kamsties and Lott[16]; however, we support the results of Basili and Selby[15] in this case.

The techniques differed in case of fault isolation efficiency where code reading performed well as compared to *dynamic testing techniques*. Our results in this case also do not support results of the previous studies. This variance of results in case of efficiency demonstrates the need for further replications to evaluate the relative efficiency of the techniques and its dependence on various factors, like nature of the program.

The type of program had significant effect in all cases, except in case of fault isolation time and fault isolation efficiency that clearly indicates that the nature of the program affects the performance of the testing techniques.

Neither order nor subjects had any significant effect on the effectiveness or efficiency. In case of fault types, all techniques were equally effective in terms of detection and isolation for all types of fault classes and types except for cosmetic faults. One important thing that emerged out of this study is that no technique was able to uncover and isolate all the defects of any kind, which supports the claim of many researchers that there is no silver bullet technique for software testing. The observed differences in effectiveness by fault class among the techniques suggest that a combination of the techniques might surpass the performance of any single technique, which was also suggested by earlier experiments like.[16,17]

Finally, we compared the empirical results of this paper with the results of Kamsties and Lott[16] experiment. We agree with the results obtained by Kamsties and Lott in most of the cases, especially that there is no difference between relative effectiveness of testing methods, and their effectiveness is influenced by the nature of the program.

The results obtained by this kind of empirical studies add to the efforts to provide empirical evidence that will help practitioners decide which technique to apply under a given condition and may facilitate them in improving software testing process. The long-term goal of such a work is to move software engineering from a craft towards an engineering discipline.[16] In addition, we believe that few experiments are not sufficient to understand this phenomenon. Further replications (both dependent and independent) of this study will help in deducing some strong conclusions regarding the effectiveness and efficiency of software testing methods and its dependence on other allied factors especially the fault types. However, performing such replications in accordance with the common framework is necessary. Moreover, we should necessarily try to mitigate major validity threats by using programs, faults, and subjects, which truly represent current software engineering practice. The programs listed in SIR repository[28] do host many programs, which truly represent the current software engineering practice and thus can be used in future experiments.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their critical assessment of the manuscript, detailed insights, and valuable and encouraging comments that enabled us to significantly improve the quality of this paper.

The authors are grateful to Mudasir Wani for his help in the running of the experiment. We also wish to acknowledge the work of Erik Kamsties and Chris Lott, especially their fine example of a replication package. The authors are also grateful to Tanja Vos, Professor, Department of Computer Systems and Computation, Universidad Politecnica de Valencia, Spain, for giving valuable suggestions. The work was partially supported by a University Grants Commission (UGC) BSR Start-Up Grant, (F.30-114/2015 (BSR)).

REFERENCES

1. Basili V. The role of controlled experiments in software engineering research. Empirical software engineering issues. Critical assessment and future directions. *Lect Notes Comput Sci*. 2007;4336:33-37.
2. Rombach H, Basili V, Selby R. Experimental software engineering issues: critical assessment and future directions. International workshop, Dagstuhl Castle, Germany, September 14-18, 1992, Proceedings, volume 706. Springer; 1993.
3. Briand LC. A critical analysis of empirical research in software testing. First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007.
4. Condori-Fernandez N, Vos T. PANEL: successful empirical research in software testing with industry. In: Proceedings of the Industrial Track of the Conference on Advanced Information Systems Engineering 2013 (CAiSE'13) co-located with 25th International Conference on Advanced Information Systems Engineering; June 21, 2013; València, Spain.
5. Juristo N, Moreno AM, Vegas S. Reviewing 25 years of testing technique experiments. *Empir Software Eng J*. 2004;9(1/2):7-44.
6. Moreno AM, Shull F, Juristo N, Vegas S. A look at 25 years of data. *IEEE Software*. 2009;26(1):15-17.
7. Juristo N, Gómez OS. Replication of software engineering experiments. In: *Empirical Software Engineering and Verification*. Berlin Heidelberg: Springer; 2012:60-88.
8. Kitchenham B. The role of replications in empirical software engineering—a word of warning. *Empir Software Eng*. 2008;13:219-221.
9. Shull F, Singer J, Sjøberg DI. Guide to advanced empirical software engineering. © Springer 2008
10. Juristo N. Towards understanding replication of software engineering experiments. *2013 ACM / IEEE Int Symp Empir Software Eng Meas*. 2013; 4(4):10-11.
11. Mendonça MG, Maldonado JC, de Oliveira MCF, et al. A framework for software engineering experimental replications. 13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS); IEEE, March 2008;203-212.
12. Shull FJ, Carver JC, Vegas S, Juristo N. The role of replications in empirical software engineering. *Empir Software Eng*. 2008;13(2):211-218.
13. Hetzel W. An experimental analysis of program verification methods [Ph.D. Dissertation]. The University of North Carolina at Chapel Hill. AAI7702047.
14. Myers G. A controlled experiment in program testing and code walkthroughs/inspections. *Commun ACM*. 1978;21(9):760-768.
15. Basili V, Selby JR. Comparing the effectiveness of software testing strategies. *IEEE Trans Software Eng*. 1987;SE-13(12):1278-1296.
16. Kamsties E, Lott C. An empirical evaluation of three defect-detection techniques. Software Engineering ESEC'95; 1995: 362-383.
17. Roper M, Wood M, Miller J. An empirical evaluation of defect detection techniques. *Inform Software Tech*. 1997;39(11):763-775.
18. Juristo N, Vegas S. Functional testing, structural testing and code reading: what fault type do they each detect? In: *Empirical Methods and Studies in Software Engineering*. Berlin Heidelberg: Springer; 2003:208-232.
19. Juristo N, Vegas S, Solari M, Abrahao S, Ramos I. Comparing the effectiveness of equivalence partitioning, branch testing and code reading by stepwise abstraction applied by subjects. Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, IEEE; 2012.
20. Lott C, Rombach H. Repeatable software engineering experiments for comparing defect-detection techniques. *Empir Software Eng*. 1996;1(3): 241-277.
21. Jedlitschka A, Ciolkowski M, Pfahl D. Reporting experiments in software engineering. In: *Guide to Advanced Empirical Software Engineering*. London: Springer; 2008:201-228.
22. Carver JC. Towards reporting guidelines for experimental replications: a proposal. In International Workshop on Replication in Empirical Software Engineering Research, Cape Town, South Africa; 2010.
23. Basili V, Selby JR, Hutchens D. Experimentation in software engineering. Technical report, DTIC Document; 1985.
24. Judd C, Smith E, Kidder L. *Research Methods in Social Relations*. New York: Holt, Rinehart and Winston; 1991.
25. Box G, Hunter J, Hunter W. *Statistics for Experimenters: Design, Innovation, and Discovery*. New York: Wiley Online Library; 2005;2.
26. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering*. New York: Springer Science & Business Media; 2012.
27. Host M, Regnell B, Wohlin C. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empir Software Eng*. 2000;5:201-214.
28. Software-artifact infrastructure repository (SIR) for experimentation. http://sir.unl.edu/. Accessed January 11, 2016

How to cite this article: Farooq SU, Quadri SMK, Ahmad N. A replicated empirical study to evaluate software testing methods. *J Softw Evol Proc*. 2017;29:e1883. https://doi.org/10.1002/smr.1883

APPENDIX A

I. Goals, hypotheses, and theories
   A. Aspects of a goal
      1. Object of study (eg, code reading, functional testing, ...)
      2. Purpose of study (eg, compare, analyze, ...)
      3. Quality focus of study (eg, effectiveness, efficiency, ...)
      4. Point of view (eg, practitioner, experimenter, ...)
      5. Context (eg, subjects, objects, environment, ...)
   B. Hypotheses
      1. Type (eg, direct observations, context factors, ...)
      2. Expected result (ie, null and alternative hypotheses)
   C. Theories
      1. Mechanisms that predict and/or explain results
      2. Derived from beliefs or related work

II. Experiment plan
   A. Experimental design
      1. Independent variables (eg, techniques, objects, order, ...)
      2. Dependent variables (eg, defects found, time required, ...)
      3. Randomization (eg, match of subject, object, and technique)
      4. Repeated measures (eg, within-subject designs)
      5. Manipulation of independent variables (eg, full-factorial, partial-factorial, ...)
      6. Null hypotheses (eg, technique A has no effect on ...)
   B. Defect-detection techniques for source code
      1. Type (eg, reading, functional testing, structural testing, ...)
      2. Other aspects (eg, test-case development, termination criteria, ...)
   C. Objects
      1. Source-code modules (eg, length, complexity, ...)
      2. Faults (eg, number, types, interactions, ...)
   D. Subjects
      1. Selection criteria (eg, participants in a course)
      2. Experience, training, and background (eg, students, professionals, ...)
      3. Ethical issues (eg, right to withdraw, anonymity, ...)
      4. How many are required (assess power of analysis procedure)
   E. Data collection and validation procedures
      1. Online and offline collection procedures (eg, forms, videotape, counts of runs, ...)
      2. Validation approaches (eg, independent sources, interviews, ...)
   F. Data analysis procedures
      1. Significance level for inferential statistics (eg, $P < .05$)
      2. Parametric techniques
      3. Nonparametric techniques

III. Experiment procedures
   A. Training activities (eg, independent work, controlled setting, ...)
   B. Conducting the experiment (eg, time periods, data validity, ...)
   C. Giving feedback to subjects (eg, comparing expectations with results)

IV. Results

A. Data (ie, the raw data collected during the study)

B. Interpretations (ie, statements about the hypotheses)

C. Summary

The ellipsis indicates similar other items in the set not listed here.

APPENDIX B

(POWER ANALYSIS FOR ONE WAY ANOVA USING MINITAB 17)

Case 1: failure observation effectiveness

```
One-way ANOVA

Sigma=5       Alpha=0.05    Number of Levels=3

            Sample                          Maximum
SS Means     Size          Power          Difference
  24.5       18            0.9631              7
```

Case 2: fault isolation effectiveness

```
One-way ANOVA

Sigma=10     Alpha = 0.05  Number of Levels = 3

           Sample                          Maximum
SS Means    Size            Power         Difference
  112.5     18              0.9809            15
```

Case 3: failure observation time

```
One-way ANOVA

Sigma=14      Alpha = 0.05  Number of Levels = 3

           Sample                          Maximum
SS Means    Size           Power         Difference
   200      18             0.9692            20
```

Case 4: fault isolation time

```
One-way ANOVA

Sigma=20    Alpha=0.05  Number of Levels=3

           Sample                         Maximum
SS Means    Size           Power        Difference
  450       18          0.9809            30
```

Case 5: total time

```
One-way ANOVA

Sigma=20   Alpha = 0.05  Number of Levels = 3

           Sample               Maximum
SS Means    Size    Power      Difference
  312.5      18     0.9134        25
```

Case 6: failure observation efficiency

```
One-way ANOVA

Sigma=0.5  Alpha=0.05  Number of Levels=3

           Sample               Maximum
SS Means    Size       Power   Difference
   0.5       18        0.9998       1
```

Case 7: fault isolation efficiency

```
One-way ANOVA
Sigma=5  Alpha=0.05  Number of Levels=3

          Sample             Maximum
SS Means   Size    Power    Difference
  40.5      18     0.9983       9

Conclusion: The power of the test is more than the required 0.90 if the sample size is
equal to 18
```

APPENDIX C

Results of the Wilcoxon rank sum (Mann Whitney) test to evaluate significant difference between code reading (CR), functional (FT), and structural testing (ST).

Ranks				**Test Statistics Grouping Variable: Tech**	
Tech	N	Mean Rank	Sum of Ranks	Mann-Whitney U	41.500
CR	18	25.19	453.50	Wilcoxon W	212.500
ST	18	11.81	212.50	Z	−3.813
Total	36			Asymp. Sig. (2-tailed)	.000
				Exact sig. [2*(1-tailed sig.)] not corrected for ties	.000
Conclusion: Since the *P* value is less than *α* = .05, we reject the null hypothesis.					
Ranks				**Test Statistics Grouping Variable: Tech**	
Tech	N	Mean Rank	Sum of Ranks	Mann-Whitney U	49.500
FT	18	12.25	220.50	Wilcoxon W	220.500
ST	18	24.75	445.50	Z	−3.560
Total	36			Asymp. Sig. (2-tailed)	.000
				Exact sig. [2*(1-tailed sig.)] not corrected for ties	.000
Conclusion: Since the *P* value is less than *α* = .05, we reject the null hypothesis.					
Ranks				**Test Statistics Grouping Variable: Tech**	
Tech	N	Mean Rank	Sum of Ranks	Mann-Whitney U	4.500
CR	18	27.25	490.50	Wilcoxon W	175.500
FT	18	9.75	175.50	Z	-4.985
Total	36			Asymp. Sig. (2-tailed)	.000
				Exact sig. [2*(1-tailed sig.)] not corrected for ties	.000
Conclusion: Since the *P* value is less than *α* = .05, we reject the null hypothesis.					