Determining the effectiveness of three software evaluation techniques through informal aggregation

Babatunde Kazeem Olorisade [a,*], Sira Vegas [b], Natalia Juristo [b]

[a] *Mathematical & Computer Sciences Department, Fountain University, P.M.B 4491, Osogbo, Osun State, Nigeria*
[b] *Facultad Informatica, Universidad Politecnica de Madrid, Madrid, Spain*

A B S T R A C T

Context: An accepted fact in software engineering is that software must undergo verification and validation process during development to ascertain and improve its quality level. But there are too many techniques than a single developer could master, yet, it is impossible to be certain that software is free of defects. So, it is crucial for developers to be able to choose from available evaluation techniques, the one most suitable and likely to yield optimum quality results for different products. Though, some knowledge is available on the strengths and weaknesses of the available software quality assurance techniques but not much is known yet on the relationship between different techniques and contextual behavior of the techniques. *Objective:* This research investigates the effectiveness of two testing techniques – equivalence class parti-tioning and decision coverage and one review technique – code review by abstraction, in terms of their fault detection capability. This will be used to strengthen the practical knowledge available on these techniques. *Method:* The results of eight experiments conducted over 5 years to investigate the effectiveness of three techniques – code reading by stepwise abstraction, equivalence class partitioning and decision (branch) cov-erage were aggregated using a less rigorous aggregation process proposed during the course of this work. *Results:* It was discovered that the equivalence class partitioning and the decision coverage techniques behaved similarly in terms of fault detection capacity (and type of faults caught) based on the programs and fault classification used in the experiments. They both behaved better than the code reading by stepwise abstraction technique.
Conclusion: Overall, it can be deducted from the aggregation results that the equivalence class partitioning and the decision coverage techniques used are actually equally capable in terms of the type and number of faults detected. Nevertheless, more experiments is still required in this field so that this result can be verified using a rigorous aggregation technique.

1. Introduction

The importance of software verification and validation (verification) cannot be overemphasized during software development. It is the process of assuring or raising the quality standard of software or any of its components. It works by systematically studying the software to gather information about its quality [1]. The verification and validation process is highly important as put by Harrold [2], it entails examining and scrutinizing the developed product and judging if it meets the customer desired quality level. If it does, the development continues, otherwise, a rework is ordered to raise the product's quality [3,4].

Regrettably, no amount of evaluation could completely assure software developers how their product will perform until they encounter a real situation [5]. Software evaluation is thus seen as an essential tool to demonstrate to or assure the client that the software is actually functional according to customer's expectation [6]. In fact, it may not be realistic or cost effective to remove all software faults prior to product release [5]. Therefore, it is crucial for developers to be able to choose from available evaluation techniques, the one or a combination most suitable and likely to yield optimum quality results for different projects.

There are two main strategies of evaluating software:

- Static analysis
- Dynamic analysis

These two techniques differ mainly on the aspect or state of the artifact under evaluation – fixed state or operation mode respectively. Each of the two strategies is further divided into several techniques, depending on the approach used.

The static analysis techniques are commonly referred to as reviews or reading techniques. In static analysis, the product under

* Corresponding author. Tel.: +234 8120451374.
E-mail address: qasimbabatunde@yahoo.co.uk (B.K. Olorisade).

evaluation is examined and scrutinized for inconsistencies and inaccuracy directly at rest [4,7]. They are used to manually detect possible defects practically from any software artifact, e.g., design document, specification document, program code, etc. The technique is static in the sense that the hard copy or paper version of the artifact in question is scrutinized by reading through it.

The intention of the exercise is to discover as much of human errors in the software artifact as possible. Reviews are useful in that they detect and remove defects, early in the lifecycle of a product. It is estimated that 30–70% of design and code defects are detected by review techniques [7]. More so, it is easier and less costly to fix mistakes at the early stage than later when dynamic techniques are applicable. The static analysis can be implemented by several techniques based on the approach to *reading* the artifact, some of these techniques are: checklist based reading, perspective based reading, code inspection, code reading by abstraction, etc. The details of code reading by abstraction used in this work and some others can be found in [7].

Dynamic analysis or testing is used to evaluate the dynamic criteria of the product. That is, the product is executed and observed for incorrect and/or unsatisfactory behavior as judged against the expected operational quality level of the product as prescribed in the specification [1,7–10]. According to Torkar [9], testing is the process of ensuring that a certain piece of software artifact satisfies its requirements.

Testing is traditionally classified into two areas: structural testing also known as glass-box testing and functional testing known as black-box testing; both named based on how the tester viewed the software/artifact under evaluation. In white-box testing, the tester will prepare test cases with the knowledge of the program constructs (insight into the program development) while the tester only test the program (functions) as is in black-box testing without any knowledge of how it was developed but what it was meant to do.

Despite these classifications and the number of techniques available, it is still difficult today to determine when the use of a certain evaluation technique is more appropriate or advantageous than others in evaluating a piece of software.

This study analyzes and summarizes empirical study results from a series of experiments performed to compare the effectiveness and efficiency of code review by stepwise abstraction as an example of static technique and decision coverage (branch coverage) and equivalence partitioning as examples of dynamic techniques. These techniques were chosen because they are methodical (less intuitive), thus, could be consistently repeated with little variation. Also, works like this will provide more insight into some of these techniques and increase their practical usage. Recent works that have used some of these techniques include [10–13]. The result of this study is significant because in order to underpin software engineering (SE), similar replications still deserve as much attention as varied replication. The result of similar replications as is the case in this study will either diminish or strengthen existing facts on the artifact under study. In this case, existing evidence on the techniques used and their comparative fault detection ability is reinforced by eight experiments used for this study.

Usually, aggregation in SE is conducted using meta-analysis techniques which are most times not suitable to SE experiments [14]. Therefore, a less rigorous technique proposed in [15] will be used. This will also contribute to the process of testing the reliability of the technique.

2. Study background

Software engineering, like other engineering fields, needs to formalize, standardize, create uniformity and have certain level of predictable functionality as well as accuracy in most of its tools, methods and procedures. For example, software evaluation techniques as earlier mentioned are broadly divisible into two – static and dynamic techniques, with each having quite a few ways of implementing them. Nevertheless, the relative or contextual benefits and drawbacks of each of the implementation techniques are still quite unknown or at best unclear [8,16,22]. Given certain circumstances of the software, there is scarce formal knowledge to guide practitioners on what evaluation technique is best applicable for optimum quality assurance.

In order to achieve this feat, researchers are aiming for extensive and exhaustive empirical research in all areas, to underpin software engineering [17], since one of the basis for development in any discipline is empirical verification of knowledge [18,19]. Software engineering researchers and practitioners are now taking advantage of empirical research, to validate their findings and work. Though, the popular strategy still involves (quasi –) controlled experiment and case study [20]. Over the years, the quality of the average empirical study in software engineering is increasing. Empirical study education (theory and practical) as it applies to software engineering is growing among researchers, consequently, the discipline is witnessing increasingly more comprehensive studies conducted on increasingly realistic programs and processes [21]. Most especially, there have been several experiments on software testing techniques as collated by [8,16,22]. Specific guidelines on and introduction to conducting experiments in software engineering are discussed in [17,18,23,24].

In line with this, with respect to verification and validation techniques, [25] initiated a study which ran through 1982, 1983 and 1984; the root of which is traceable to [26] and [27]. The experiment studied the effectiveness and efficiency of different software evaluation techniques. This experiment was replicated by Kamsties and Lott [28] as well as [29]. The summary of these experiments is presented in Table 1.

This paper follows in line, to contribute to advancing the available pragmatic knowledge on software evaluation techniques. It focus on analyzing and summarizing data from a series of experiments performed to compare the effectiveness of code review as an example of static technique and decision coverage and equivalence partitioning as examples of dynamic techniques.

A group of researchers at the Universidad Politécnica de Madrid (UPM) in Spain initiated a series of experiments, designed to achieve the same aim as Basili and Selby [25], Juristo and Vegas [30] – investigate the effectiveness and efficiency of three software evaluation techniques. The root of these studies is traceable to [26,27]. The materials used for the experiments was extracted from Kamsties and Lott's experiment [28,30], as well as Wood et al. [29]. The experiments used for this study are eight and it spans four sites – UPM, Universidad Politécnica de Valencia (UPV), Spain, Universidad ORT Uruguay and Universidad de Sevilla (UdS), Spain. These experiments were conducted and analyzed by different researchers. Thus, there is a need for a joint analysis of all the experiments.

2.1. Hypothesis and response variables

The main aim of these series of experiments is to investigate the hypothesis whether or not the effectiveness of code evaluation techniques has anything to do with the fault types present in the program. Therefore, the **response variable** is *effectiveness* which is measured in terms of the number of subjects that detects a given fault for each fault in the program.

There are three basic **factors** used, *technique, fault type* and *program type*. However, since the program size was considered small, a fourth factor – *version* was introduced to increase the number of faults seeded into the programs.

Table 1
Previous studies comparing functional, structural testing and code reading techniques.

Study	Type	Technique	Purpose	Result[a]
Hetzel, 1976	Experiment	Functional test, structural test and inspection	Detection	Effectiveness: testing > inspection
Myers, 1978	Experiment		Detection	Effectiveness: inspection = testing; complementary, but different for some classes of defects
Masili and Selby, 1987	Experiment		Detection	Effectiveness and efficiency depends on software type
Kamsties and Lott, 1995	Experiment (Replication)		Detection and isolation	Effectiveness: no significant difference Efficiency: testing > inspection
Roper et al, 1997	Experiment		Detection	Effectiveness: no significant difference Efficiency: testing > inspection combination better

[a] "$x > y$" means x is better than y.

To investigate how the combination of these factors influence fault detection, using each of the techniques. The experiments' **hypotheses** were set as below [30]:

H_{01}. The fault detection technique, program and fault type do not impact effectiveness.

H_{11}. The fault detection technique, program and fault type has an impact on effectiveness.

H_{02}. The *failures generated by the faults* have no impact on its visibility.

H_{12}. The *failures generated by the faults* have an impact on visibility.

H_{03}. The fault detection technique, program and fault type do not impact efficiency.

H_{13}. The fault detection technique, program and fault type has an impact on efficiency.

2.2. Factors

The factors for the experiments are: fault type, program type, technique and version. Each is discussed in more details below:

- **Fault types:** The fault classification used in the experiments is based on the classification used by Basili in his experiment [25]. The introduction of program version facilitated the elimination of fault repetition, thus, each program contained seven distinct faults. The faults are majorly classified into two – omission (something missing) and commission (something incorrect). The seven faults used are:
  - F1: Cosmetic, omission: Where an error message is expected but there was none.
  - F2: Cosmetic, commission: Faults where there are spelling mistakes in messages.
  - F3: Initialization, omission: In cases where a data structure is not initialized.
  - F4: Initialization, commission: In cases where a data structure is wrongly initialized.
  - F5: Control, commission: When a conditional statement is given an incorrect predicate (logical expression(s)).
  - F6: Control, omission: When a predicate is completely missing but expected.
  - F7: Computation, commission: Incorrect arithmetic operation, e.g. an incorrect arithmetic operator on the right hand side of an assignment statement.
- **Technique:** The techniques for the experiments are: code review by stepwise abstraction for the static technique, equivalence partitioning for the functional technique and branch coverage for the structural technique. The dynamic techniques were used in the following manner:
  - The subjects applied the technique to generate test cases. These test cases were later analyzed to determine what faults they can detect.
  - The subjects were given test cases supplied by the experimenters to execute. This was to eliminate any bias in the earlier results obtained on the structural and the functional techniques.
- **Program:** Three programs written in C language were used. They are – *cmdline*, *nametbl* and *ntree* – these three came with the original experimental material used for the experiments [25]. The programs' sizes are 308, 342 and 271 Lines of Code (LOC) including white space respectively.
- **Version:** Each program was seeded with the same number of faults (seven). Due to the fact that the program size is small, not much fault could be inserted into each. Therefore, two versions of each program marked by the fault they contain were produced for the experiments.

2.3. Experimental design

The experiments were conducted using *Cross-over design*. In this design, each subject will apply all the three techniques. The design is presented in Table 2 below. The subjects were in a group of 7 or 8. Each group was randomly assigned a program and a technique but the group members worked independently.

In Table 2, 'CR' stands for code review, 'S' for structural technique and 'F' for functional technique. Apparently, all the groups exercised with all the techniques, one technique on one program executed as shown in Table 3. Though, due to a few constraints, some variations to this design exist in some of the replications. The variations are not expected to affect the results in anyway.

The techniques are applied within the following context:

- Equivalence class partitioning: The subjects are supplied with the specification document from where the subjects identify the desired inputs necessary for the program. The subjects have

Table 2
Experiment design for experiment.

Program	cmdline			ntree			nametbl		
Technique	CR	S	F	CR	S	F	CR	S	F
Group 1	X	–	–	–	X	–	–	–	X
Group 2	X	–	–	–	–	X	–	X	–
Group 3	–	X	–	–	–	X	X	–	–
Group 4	–	X	–	X	–	–	–	–	X
Group 5	–	–	X	X	–	–	–	X	–
Group 6	–	–	X	–	X	–	X	–	–

the task to partition each input into valid and invalid classes [36]. Valid and invalid equivalence classes are identified according to a set of predefined heuristics based on the expectation from the specification – a condition, range of values, certain number, or command. In a case where input values are not identically handled, the input will be further splitted before being partitioned.
- Afterwards, test cases are developed based on the equivalence classes identified. As many as possible test cases are written until all uncovered valid classes are covered. Each invalid class is also taken in turn until test cases have been written for all of them.
- Decision coverage: This is also known as branch testing. The approach adopted in this study is to ensure that each branch alternative is exercised at least once with attempt to cover 100% branch coverage.
- Code reading by stepwise abstraction: Subjects are given the code to study and attempt to create an abstraction that represents the program summary – possible outcomes, irrespective of its internal control structure and data operations. This preliminary abstraction is continuously built upon until the whole code is covered. Then, subjects compare the program specification with their abstraction, to identify inconsistencies between their abstraction and expected program behavior.

2.4. Experimental subjects

The subjects used in all the experiments are final year computer science students who are given at least 4-h (up to 16 h at UPM) training on the tasks of the experiment.

2.5. Experiment and replication execution

At UPM, the experiments were organized in three different sessions as shown in Table 3, the reason for the experiments was clear to all the subjects and they are also aware that the result will form part of their performance grading. The students were handed required documentation and asked to study them, the training for the exercise was given as part of a course. No student had a prior knowledge of what technique or program they were going to work with before the experiment.

However, the experiment was adapted based on the situation of each of the replication sites. The constraints encountered are highlighted below and the adaptation shown in following tables.

- **UPV Experiment:** In order to conduct the same experiment at the UPV, all documentations used for the UPM experiment were transferred to the researchers at the UPV. Additionally, a series of meetings took place between the researchers of both schools to iron out gray areas and facilitate similarities in the replication of the experiments. Nevertheless, a few environmental constraints enforced some alteration in the design and execution of the experiment compared to the UPM's package. The study objectives, hypothesis and response variable as well as factors and alternatives were the same as that of the UPM.

The following differences existed between the UPV environment and UPM environment as far as the experiment is concerned:

- Subjects are already acquainted with the techniques
- Less time was available to perform the experiment
- Less time available per session
- Training and execution of the experiment cannot be sequential

These four constraints lead to the four new conditions as shown in Table 4.

Training was organized in the form of a refresher tutorial on the techniques in form of a practical exercise; this was merged with experiment execution.

- **UdS Experiment:** As in the case of the UPV, the same set of documentation, used for the UPM experiments was transferred to researchers at UdS. Also, a series of meetings and discussions took place between the researchers at UPM and UdS to explain any ambiguity in the materials to enhance understanding. The unique environment of the UdS also forced the researchers to alter the UPM design to adapt it to their own environment. Apart from the environmental constraints, other things like the experiment objectives, hypothesis, factors and response variables remained as it was in UPM.

Five conditions were different between the UPM and UdS environment regarding this experiment. These are:

- Less time available to run the experiment
- Less time available per session of the experiment
- Training and operation not sequential
- Subjects were already acquainted with the techniques
- Insufficient computer systems

The summary of conditions and changes is shown in Table 5 below:

- **ORT Experiment:** Similar documentation transfer and explanation applies to ORT as to UPV and UDS.

In this case, there were three constraints:

- Less time available
- Computer room not available
- Subjects are junior with no programming experience

The constraints forced the changes as highlighted in Table 6.

Table 3
Experiment execution for experiment.

Day Program	Day 1 cmdline	Day 2 ntree	Day 3 nametbl
Group 1, Group 2 Group 3, Group 4 Group 5, Group 6	Review Structural Functional		
Group 4, Group 5 Group 1, Group 6 Group 2, Group 3		Review Structural Functional	
Group 3, Group 6 Group 2, Group 5 Group 1, Group 4			Review Structural Functional

3. Aggregation

Though, this research is to present the summary (aggregate) result of the eight experiments conducted to investigate the effectiveness of three software fault detection techniques; at this stage, it was not intended to a formal aggregation method (meta-analysis) because the experiments are not sufficient enough for a reliable result [14]. Therefore, a qualitative deduction approach (informal aggregation) described in details in [15,35] was used to systematically synthesize the results.

The method – informal aggregation, as proposed consists of four steps:

Table 4
Environmental differences and adjustment in the UPV experiment.

New condition	Change on experiment
Less time available	Code review technique left out Test cases executed for just one of the programs
Two hours per session	Test case generation will be performed in one session and test case execution in another session
Training and operation not sequential	One technique applied on all three programs in one session after the training on such technique
Subjects already acquainted with techniques	Refresher tutorial of 4 h rather than 16 h of lessons with the same material

Table 5
Environmental differences and adjustment in the UdS experiment.

New condition	Change on experiment
Less time available	Test cases executed for just one of the programs
Two hours per session	Test case generation will be performed in one session and test case execution in another session
Training and operation not sequential	One technique applied on all three programs in one session after the training on such technique
No sufficient computer systems	Subjects work in pairs
Subjects already acquainted with techniques	Refresher tutorial of 4 h rather than 16 h of lessons with the same material

Table 6
Environmental differences and adjustment in the ORT experiment.

New condition	Change on experiment
Less time available	Code review technique left out Experiment ran in one session One of the programs left out
Computer room not available	Test case execution left out
Junior subjects without programming language experience	Junior subjects

- **Extraction:** Extract the significance value of all the treatments from the various ANOVA tables and summarize them in one table.
- **Classification:** Classify the patterns using some code (e.g., alphabets).
- **Classification Ranking:** Study the distribution pattern in the code table and rank the pattern.
- **Deduction:** Study each category and qualitatively deduce evidence from each.

Details of each of these steps are fully described in [15,35]. The result of applying each step in this study is presented in Sections 3.1–3.4.

3.1. The extraction step

As suggested in [15,35], we decided to employ the flexible combination approach. This is because each of our experiment was analyzed using different confidence level limit. We decided to tolerate the 99% and 95% confidence limit. The result of this step is presented in Table 7. Black colored cell indicates significance values significant at 99% limit while the grey cells represents the values significant at 95% limit while the white cells are not significant at both confidence levels.

3.2. The classification step

The classification procedure follows the description given in [15,35] and the result is shown in Table 8. In this case, two confidence levels were accommodated thus, $x = 3$ while $y = 7$.

3.3. The classification ranking

After the classification, the next step is to rank (numerically) the classification table based on the defined strength or clarity of the knowledge presented by the combination in each column (the row entry) [15]. There are three sub-steps to follow in this case.

Table 7
Compressed ANOVA table for the experiments.

Notes	Treatments	Significance							
		UPM 01	UPM 02	UPM 03	UPM 04	UPM 05	UPV 05	ORT 05	UdS 05
Model used: Type III Sum of Squares Significance level: 0.01 and 0.05	Program	0.025	0.390	0.032	0.907	0.002	0.026	0.003	0.006
	Technique	0.000	0.000	0.000	0.000	0.000	0.764	0.003	0.000
	Version	0.256	0.095	0.114	0.323	0.361	0.388	0.635	0.057
	Fault	0.062	0.316	0.853	0.009	0.215	0.013	0.000	0.050
	Program * Technique	0.006	0.017	0.001	0.048	0.040	0.022	0.014	0.076
	Program * Version	0.145	0.590	0.797	0.156	0.061	0.054	0.710	0.832
	Program * Fault	0.000	0.041	0.056	0.002	0.000	0.021	0.001	0.018
	Technique * Version	0.076	0.539	0.440	0.904	0.236	0.039	0.221	0.025
	Technique * Fault	0.001	0.269	0.028	0.472	0.028	0.255	0.265	0.033
	Version * Fault	0.522	0.152	0.046	0.361	0.670	0.014	0.092	0.087

Table 8
Code creation based on combination of different possibilities.

Code	Significant at 0.01	Significant at 0.05	Not significant
A	No	No	Yes
B	No	Yes	No
C	No	Yes	Yes
D	Yes	No	No
E	Yes	No	Yes
F	Yes	Yes	No
G	Yes	Yes	Yes

Table 9
Ranking of the various codes.

Code	Significant at 0.01	Significant at 0.05	Not significant	Rank
A	No	No	Yes	1
B	No	Yes	No	3
C	No	Yes	Yes	4
D	Yes	No	No	1
E	Yes	No	Yes	4
F	Yes	Yes	No	2
G	Yes	Yes	Yes	4

3.3.1. Assignation

At this stage, each row of Table 8 is ranked according to the clarity of evidence or weight of the evidence presented by the pattern in the row. The result from our study is shown in Table 9.

The ranks in Table 8 are fully explained in [15,35]. For this study the classification range [15,35] are: a significant or non significant situation that lies between 0% and 29% presents no tendency, 30–40% we do not know what to say, 40–60% there seems to be a tendency and 60% upward there is a tendency.

3.3.2. Annotation

The summarized ANOVA (Table 6) produced in step 1 is then interpreted with corresponding codes and ranks as shown in Table 10.

3.3.3. Streamlining

In this study, there are eight experiments and we decided to use the ordinal scale (significant, significance tendency, ambiguous and not significant) as proposed in [15,35]. The output from this process is shown in Table 11.

The status column in Table 10 is the final aggregated deduction (decision) on the treatment in question. This is decided based on the (accumulated) relative percentage of the number of experiments found sub-columns under the "Number of experiments significant at:" column (Table 11).

3.4. The deduction step

From Table 11, it has become clear that the null hypothesis will be accepted for 'version and its interactions' but will be rejected (generally) for technique, program, fault and their interactions. Therefore, it implies that from these experiments, we may establish with some level of confidence that the effectiveness of a technique may be influenced by the fault type and program type.

4. Analysis

The analysis in this section is a reflection and discussion on the result shown in Table 11. This discussion will draw on the statistical analysis result produced from SPSS for each of the treatment. This study only provides summary of the SPSS results that have been compacted for visual comparison. The extensive individual experiment's statistical data and SPSS analysis graphs like profile plot and stock plot can be found in [15,35] and from the Empirical Software Engineering research group (GriSE[1]) at UPM.

The treatments will be discussed based on the classification from Table 10.

4.1. Significance status

The technique is the only treatment categorized as *significant* (Table 11). Seven out of eight of the experiments are significant at 0.01 significance level but because 'technique' is a main effect it will be analyzed further.

In Fig. 1, the result of the separate profile plots are pooled together for compactness and visual comparison. From the figure, it is apparent that the structural technique exhibits the best behavior with all the experiments recording mean value greater than 75 except ORT 05 with mean value of 71. The functional technique had five experiments with mean values above the 75 line mark, two above 70 but less than 75 and one above 65 but less than 70. All experiments record mean values below 50 for the code review technique except UPM 04 and 05.

From this, it could be said that the structural and the functional technique both display the tendency of behaving almost equally and performing better than the code review technique.

4.2. Significant tendency

4.2.1. Program

1. The summary of the SPSS profile plot is shown in Fig. 2, no definite or regular pattern is discernible across all the experiments from the figure. Nevertheless, four possible different patterns can be deduced:
2. All programs behaved the same: This is the situation with the UPM 02 and UPM 04 experiments. All the programs in the experiments exhibit similar behavior, i.e. no remarkable difference was found in the number or type of faults each detect (effectiveness). However in all the cases, 'nametbl' ended up having highest mean value. This happens for the experiments where program was not significant.
3. Nametbl' and 'ntree' behaved similarly: In this situation both behaved better than the 'cmdline' program. Examples are in UPV 05 and UdS 05 experiments. UPV 05 is significant at 0.05 while UdS 05 is significant at 0.01.
4. 'Ntree' behaved better than 'nametbl' and 'cmdline': The last situation is when 'ntree' behaved better than the two other programs which behaved similarly. This happens in UPM 05 which is significant at 0.01.

Summarily, the pattern presented in Fig. 2 shows that 'nametbl' exhibited the best behavior – seven of the experiments have mean values greater than 65. Though, this result interpreted in isolation, may not be so much reliable as there are interaction effects involving 'program' and other factors in all the experiments.

It is interesting to note the fact that the result of UPM 05 exhibit a behavior which is somewhat different to the pattern noticed in other results. 'Nametbl' has the lowest value and 'ntree' the highest, which does not happen in any other experiment. This may probably be due to the influence of the order of program execution either in the experiment or other experiments. ntree and cmdline exhibited behavior similar to each other, with most of the results within the 65–75.

[1] www.grise.upm.es.

Table 10
Treatments with assigned codes and ranks.

Notes	Treatments	Significance								Code	Rank
		UPM 01	UPM 02	UPM 03	UPM 04	UPM 05	UPV 05	ORT 05	UdS 05		
Model used: Type III Sum of Squares	Program	0.025	0.390	0.032	0.907	0.002	0.026	0.003	0.006	G	4
	Technique	0.000	0.000	0.000	0.000	0.000	0.764	0.003	0.000	E	4
	Version	0.256	0.095	0.114	0.323	0.361	0.388	0.635	0.057	A	1
Significance levels: 0.01 and 0.05	Fault	0.062	0.316	0.853	0.009	0.215	0.013	0.000	0.050	G	4
	Program * Technique	0.006	0.017	0.001	0.048	0.040	0.022	0.014	0.076	G	4
	Program * Version	0.145	0.590	0.797	0.156	0.061	0.054	0.710	0.832	A	1
	Program * Fault	0.000	0.041	0.056	0.002	0.000	0.021	0.001	0.018	G	4
	Technique * Version	0.076	0.539	0.440	0.904	0.236	0.039	0.221	0.025	C	4
	Technique * Fault	0.001	0.269	0.028	0.472	0.028	0.255	0.265	0.033	G	4
	Version * Fault	0.522	0.152	0.046	0.361	0.670	0.014	0.092	0.087	C	4

Table 11
Status of each treatment.

Code	Rank	Treatment	Numbers of experiments significant at:			Status
			0.01	0.05	Not significant at both	
A	1	Version	0/8	0/8	8/8	Not significant
	4	Program * version	0/8	0/8	8/8	Not significant
C	1	Technique * version	0/8	2/8	6/8	Not significant
	4	Fault * version	0/8	2/8	6/8	Not significant
E	4	Technique	7/8	0/8	1/8	Significant
G	4	Program	3/8	3/8	2/8	Significant tendency
	4	Fault	3/8	1/8	4/8	Ambiguous
	4	Program * fault	3/8	4/8	1/8	Significant tendency
	4	Technique * fault	1/8	3/8	4/8	Ambiguous
	4	Program * technique	2/8	5/8	1/8	Significant tendency
Total			19/80	20/80	41/80	

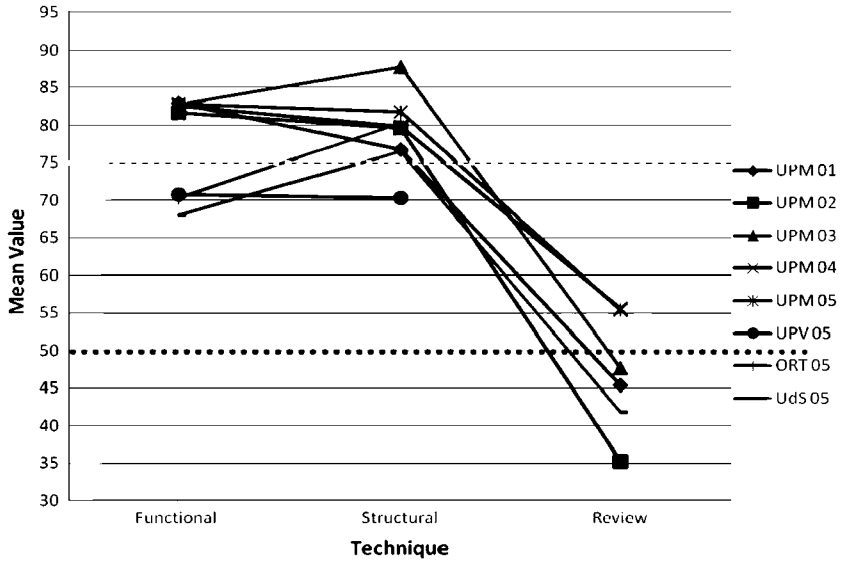

Fig. 1. Technique performance for all experiments based on mean values.

4.2.2. Program–fault interaction

The pattern deduced from the profile plot for most of the experiments for this interaction display a disordinal relationship [31]. According to Hair, it is recommendable to redesign and run the experiments again until the profile plots exhibits an ordinal relationship or the interaction found not significant. Nevertheless, we will attempt to analyze the interaction for the sake of this study.

Seven of the eight experiments have a significant interaction effect between 'program' and 'fault'. Four of the programs (UPM 01, UPM 04, UPM 05 and ORT 05) are significant at 99% confidence level while UPM 02, UPV 05 and UdS 05 are significant at 95% confidence level. The behavior of the faults varied in the different programs.

The summary is presented in Table 12, each column in the table depicts the behavior of each fault with respect to the different programs for each experiment. Table 13 shows the mean, range, maximum value and the lowest value for each fault in each program for all the experiments.

Using these tables, the following can be infered:

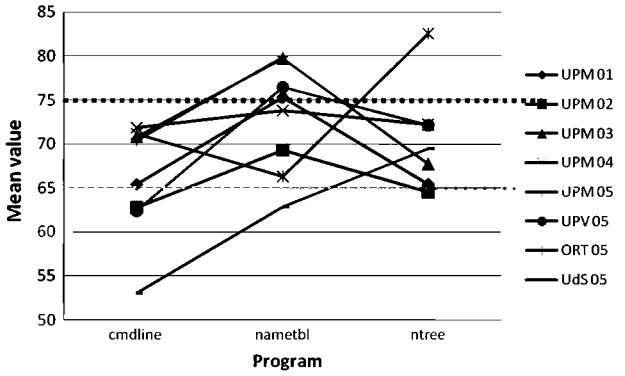

Fig. 2. Program performance for all experiments based on mean values.

- F2 and F6 summarily behaved very well and closely similar in all the programs. This fact is supported with the closeness in the low range values and fairly high mean values.
- F4 records wide variability across the programs. Its behavior is also not impressive. Though, it is a bit high in nametbl.
- F3 record high values in all the programs but with wide notable variability.
- F1 and F5 also exhibited wide variability across all programs with an average performance.
- F7 behaved closely the same in cmdline and nametbl better than ntree.

The overall average behavior of all the faults in the programs can be summarized as shown in Table 14 below.

4.2.3. Program–technique interaction

Seven of the eight experiments showed significant effect with the program–technique interaction. UPM 01 and UPM 03 were significant at 99% confidence level while the rest were significant at 95%. Tables 15 and 16 is a summary of the techniques' behavior with respect to each program. It was clear from the table that:

- The functional and structural technique behaved the same, better than code review irrespective of the program for all experiments.
- There is variability in the techniques' performance depending on the program.
- Code review behaved best with 'ntree' with a tendency of matching the dynamic techniques' performance for the program but worst with 'cmdline' in all the experiments.
- Functional and structural techniques behaved best with 'nametbl' and worst with 'ntree' except in UPM 05. This deviation can be considered as negligible.

4.3. Not clear status

Fault and fault–technique interaction are the effects that fell under this classification. Due to the fact that the analysis in this study involved eight experiments, overlooking the effects in this category may result into loss of vital knowledge concerning the factors. Therefore, the pieces of knowledge deductible from the effects in this section will be used together with those of the effects in the *significant* and *significant tendency* categories to have a full grasp of knowledge pieces passed along by all the experiments.

4.3.1. Fault

Two of the experiments are significant for fault at 0.01 level of significance, these are UPM 04 and ORT 05, also two are significant at 0.05, UPV 05 and UdS 05 and the rest are not significant. Fig. 3 presents the profile plot for the different experiments in a single graph. No definite or consistent pattern was visible from the fault's profile plots or Fig. 3 for all the experiments.

Nevertheless, the facts below were deducted from the plots and cluster distribution of the mean values as representative of all the experiments.

- F3 maintains a high value in all the experiments. Actually, it was the fault with highest value in 6 out of the eight experiments. It shows a very good visibility tendency.

Table 12
Program–fault interaction organized according to each fault.

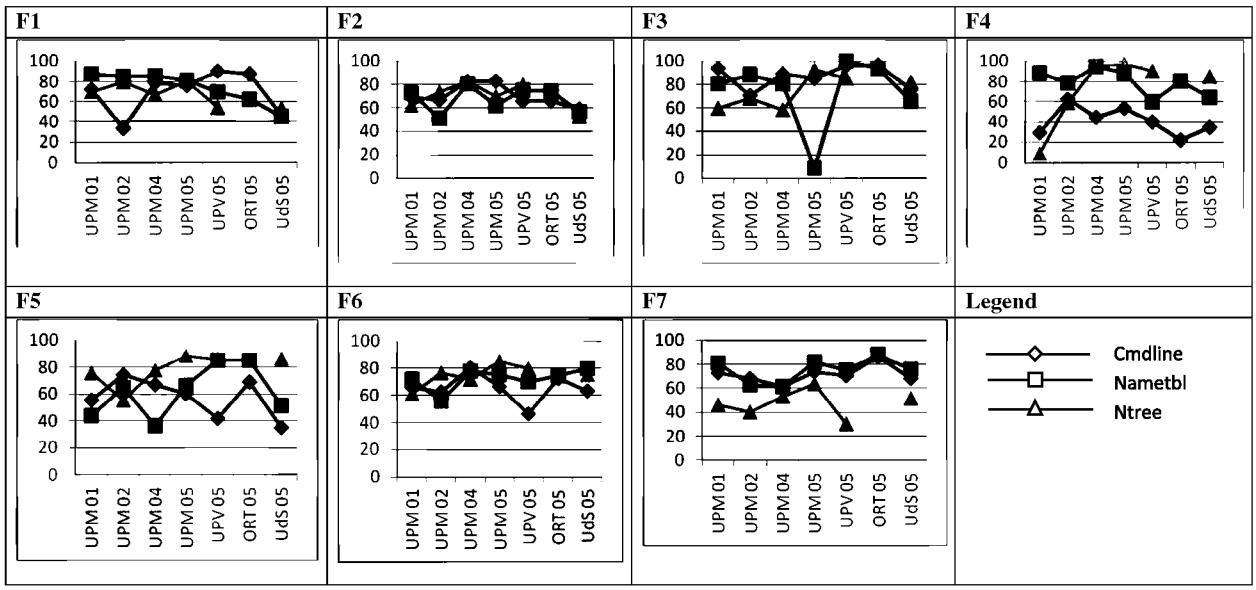

Table 13
Central tendency measures for program–fault interaction values.

	Cmdline				Nametbl				Ntree			
	Mean	Range	Max.	Min.	Mean	Range	Max.	Min.	Mean	Range	Max.	Min.
Fl	68.98	56.68	90.00	33.32	73.96	41.97	87.50	45.53	67.27	26.42	80.00	53.58
F2	70.42	24.42	83.33	58.91	67.57	29.77	80.56	50.79	70.24	30.46	83.33	52.87
F3	86.80	25.94	96.77	70.83	74.05	91.70	100.00	8.30	74.31	35.00	93.33	58.33
F4	57.45	39.54	74.64	35.10	61.73	49.18	85.29	36.11	78.33	33.57	88.89	55.32
F5	40.90	40.11	62.56	22.45	79.36	34.44	94.44	60.00	72.26	88.40	97.22	8.82
F6	65.50	33.89	80.56	46.67	72.17	24.57	80.09	55.52	74.94	24.09	85.00	60.91
F7	71.53	25.00	86.11	61.11	75.03	27.13	88.24	61.11	47.40	33.89	63.89	30.00

Table 14
Overall average behavior of program–faults interaction.

Fault behavior (average of marginal means)	Cmdline	Nametbl	Ntree
Very good ($\geqslant 75$)	F3	F5, F7	F4, F6
Good ($65 \leqslant 74$)	F1, F2, F6, F7	F1, F2, F3, F6	F1, F2, F3, F5
Average ($55 \leqslant 64$)	F4	F4	–
Poor ($\leqslant 54$)	F5	–	F7

Table 15
Performance of each technique in the different programs.

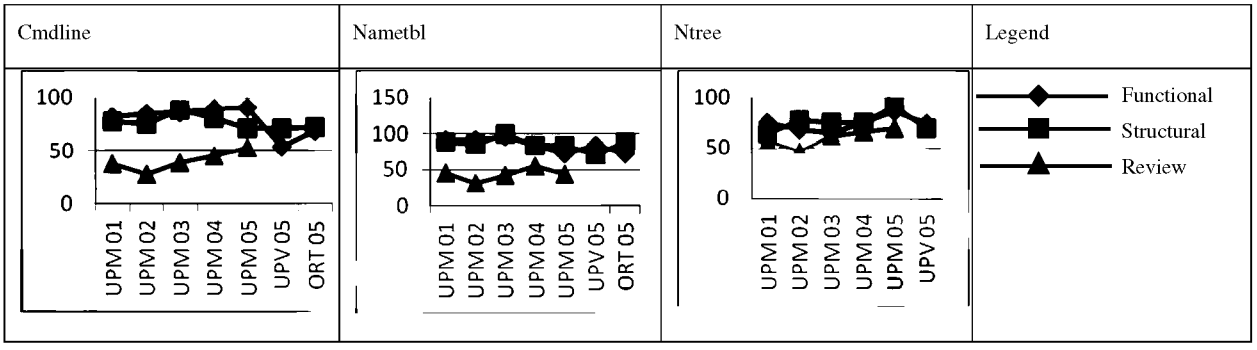

Table 16
Central tendency measures for the program–technique interaction.

	Functional				Structural				Review			
	Mean	Range	Max.	Min.	Mean	Range	Max.	Min.	Mean	Range	Max.	Min.
Cmdline	79.09	36.06	89.91	53.85	76.53	17.22	87.9	70.68	40.42	24.93	52.86	27.93
Nametbl	84.40	24.94	96.79	71.85	85.51	30.00	100	70.00	43.36	24.05	54.76	30.71
Ntree	74.22	21.00	86.48	65.48	75.69	27.14	91.43	64.29	60.43	22.48	69.76	47.28

- Faults F1 and F6 maintained the same level of performance across all the experiments. These faults also exhibit good tendency for visibility.
- F2, F5, F7 and F4 showed fairly good tendency for visibility.

4.3.2. Fault–technique

UPM 01, UPM 03, UPM 05 and UdS 05 are the four experiments that showed significant effect for fault-technique interaction. All the experiments are significant at 95% confidence level except for the UPM 01 experiment that is significant at 99%.

On the other hand, Table 17 showed the behavior of the faults with respect to the techniques while Table 18 presents some measure of dispersion for each fault's interaction with technique. This behavior is summarized in Table 19.

The pattern of behavior as highlighted below can be discerned from the tables:

- In general, functional technique behaved same as the structural technique and both are better than the code review for all faults.
- There is variability in the performance of all the techniques based on fault. Which implies that some faults might be visible than others.
- The code review technique displays a tendency of matching the performance of the dynamic techniques with faults F1, F3 and F7 which are: omission, cosmetic; omission, initialization and commission, computation faults respectively.
- The code review exhibited a consistent worse performance behavior with F2, error of commission, cosmetic.
- The dynamic techniques exhibited good performance behavior for all fault types in all experiments except UdS 05 where the average performance is not so high occasionally.
- The code review appears not good for the errors of commission, cosmetic – F2, commission initialization – F4 and F6, but show good tendency with F5 – commission, control.

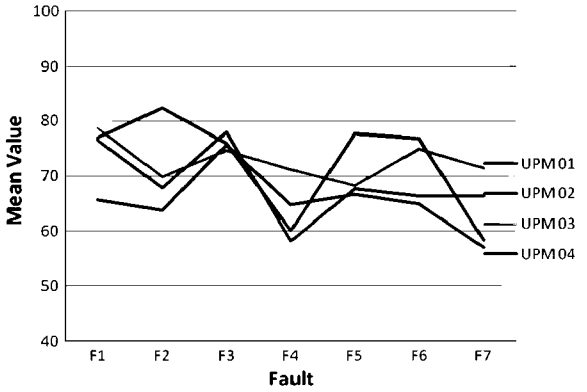

Fig. 3. (a) Aggregated behavior of fault in all the experiments and (b) aggregated behavior of fault in all the experiments.

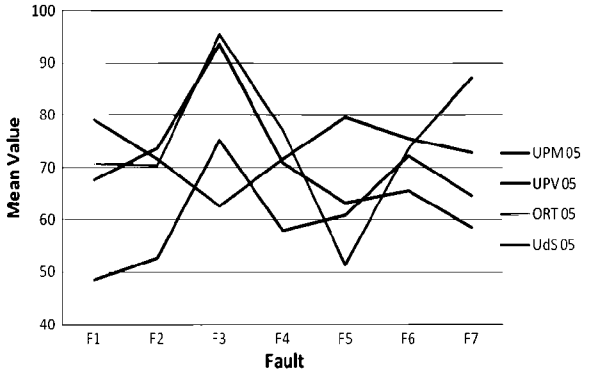

Fig. 3. (*continued*)

4.4. Focused analysis

Here is a concise presentation of the findings derivable from the analysis discussion presented above.

- Technique:
  - The equivalence class partitioning and decision coverage were more effective that the code reading by stepwise abstraction in all the experiments (that actually used the technique).
  - The equivalence partitioning and decision coverage displayed relatively equal effectiveness except in UDS 05 and ORT 05 where the functional technique's performance fell below the 75% mark and decision coverage was above it.
- Program: No regular pattern is noticed in the behavior of the program. So, nothing much can be deducted from the results on program.
- Program * Fault: The program * fault interaction exhibited a disordinal relationship. Therefore nothing much can be deduced from this interaction.
- Program * Technique:
  - Irrespective of the program the equivalence partitioning and decision coverage displayed more effectiveness than code reading by stepwise abstraction.
  - Program appears to affect the effectiveness of code reading by abstraction. Its performance vary in the ntree program is better than others.
- Fault: Fault was significant in four of the experiments – UPM 04, UDS 05, UPV 05 and ORT 05. However, the fault * technique interaction was disordinal, thus, nothing extensive can be deduced further.

5. Discussion and future direction

Eight experiments were used for this study. Five of them were conducted at UPM and one each at ORD, UdS and UPV. The statistical results of the study were aggregated using a less rigorous aggregation approach to systematically summarize the results. A joint ANOVA table was created from the individual analysis of the different experiments. For fear of loss of knowledge, the confidence interval level for significance was maintained at both 99% and 95% with 99% held as presenting more reliable evidence than 95%. This idea was used to create a significance classification for

Table 17
Performance of the techniques with each fault.

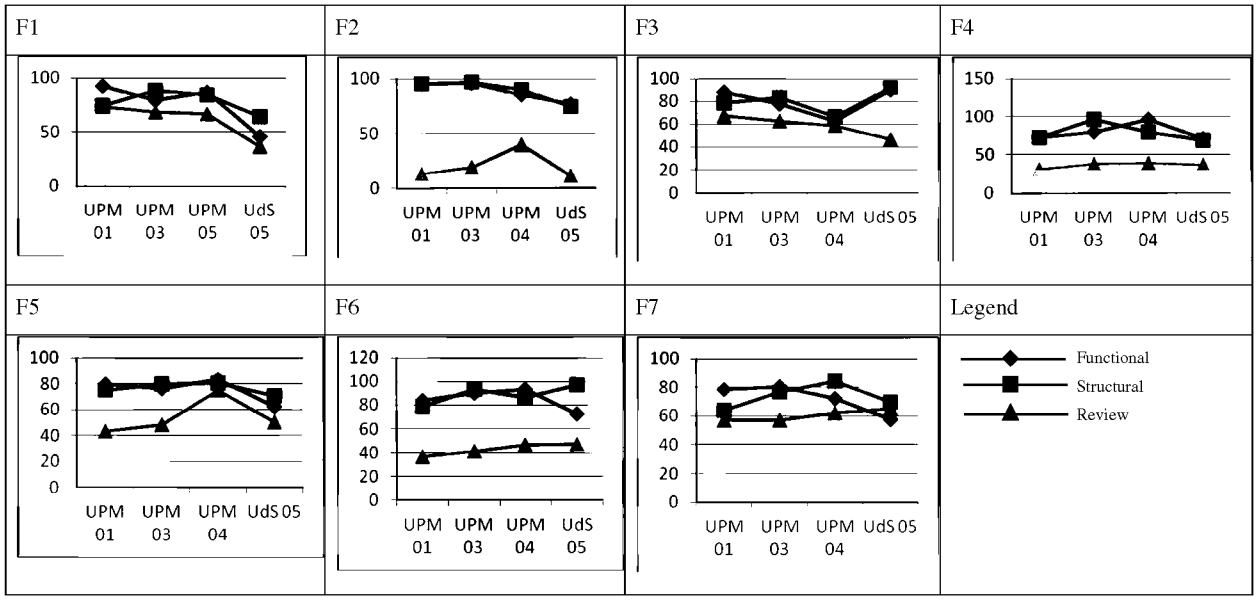

Table 18
More statistical measures for the fault technique interaction.

	Functional				Structural				Review			
	Mean	Range	Max.	Min.	Mean	Range	Max.	Min.	Mean	Range	Max.	Min.
Fl	76.03	47.65	92.91	45.26	77.52	24.73	88.43	63.70	60.86	36.87	72.92	36.05
F2	88.19	19.38	95.89	76.51	89.18	22.43	96.67	74.24	20.68	28.24	39.44	11.20
F3	79.68	27.55	90.05	62.50	80.10	25.15	91.82	66.67	58.82	21.11	67.36	46.25
F4	80.04	25.04	96.67	71.63	79.32	27.98	96.67	68.69	35.49	7.77	38.33	30.56
F5	75.43	21.29	83.33	62.04	76.52	10.05	80.56	70.51	54.18	32.08	75.00	42.92
F6	85.07	20.77	93.33	72.56	88.85	18.25	96.82	78.57	42.99	10.56	47.37	36.81
F7	72.34	22.67	80.56	57.89	73.58	20.28	84.17	63.89	60.45	8.46	65.40	56.94

Table 19
Summarized behavior of the faults with each technique.

Fault detection (average of marginal means)	Functional	Structural	Code review
High ($\geqslant 75$)	F1, F2, F3, F4, F5, F6	F1, F2, F3, F4, F5, F6	
Fairly high ($60 \leqslant 74$)	F7	F7	F1, F7
Low ($50 \leqslant 59$)			F3, F5
Very low ($\leqslant 49$)			F2, F4, F6

the effects – significant, significant tendency, unclear and not significant. The technique, was found from the exercise to be significant, the program, the program * technique and the program * fault interactions were found to have significance tendency and fault and fault * technique interaction found to have ambiguous or unclear significance.

We used the mean values, the confidence interval, the profile plot and the stock plot [15] to establish behavior of treatments in the different experiments that are found to be globally significant or have a tendency to be. Also, we decided that since this is not the analysis of a single experiment, it is better to also study the *main effects* even if there were interaction effects involving the main effect.

The approach to studying the main effect is different to that of interaction effects:

- For any significant main effect selected from step 3, all the experiments shall be studied to have a general insight into the factor's characteristics. Even if it is not specifically significant [15].
- However, for the interaction effects, those experiments that are significant at the specified confidence level(s) only were selected for further study [15].

Eventually, it was overall discovered from the eight experiments that:

- The equivalence class partitioning and decision coverage techniques behaved identically in terms of their effectiveness; which means, in the context of the programs and fault types used for the experiments none of the two can be said to be more effective than the other in terms of producing test cases that could expose the faults.
- The equivalence class partitioning and decision coverage techniques behaved better (is more effective) than code reading by stepwise abstraction.
- Combination of code reading by stepwise abstraction with either of equivalence class partitioning or decision coverage techniques is advised. This is because despite its generally worst performance, it still displayed promising effectiveness tendency with certain faults – F2, F4 and F6.
- Effectiveness of code reading by stepwise abstraction is influenced by program type.
- Fault type did not affect the effectiveness of the techniques.
- Combination of the reading techniques with either of the testing techniques may provide 100% detection of the faults present in the codes.

Though, at this stage of the study, it is unclear if subject experience has a role to play in the poor performance noticed in the effectiveness of code reading by stepwise abstraction as this contrasts earlier findings in similar researches that compared other code reading techniques to testing techniques [24,28,29]. Yet it is still in agreement with results of some studies [10,11,13,32] that used code reading by stepwise abstraction. So, future studies may look into subjects' experience as a factor or compare other review technique(s) with the functional and structural techniques used.

5.1. Threats to validity

The results of the experiments are threatened by factors as described in [23]. The threats in this case are those associated with the conduct of the individual experiments as this will have a net effect on the outcome of the experiment results. The identified threats are:

- Internal Validity: The order in which the techniques were applied may have effect on the results. Most importantly, in cases where not all the techniques were applied or not all the programs were exercised. Though, was considered to be insignificant compared to the threat posed by information sharing between the groups.
- Eternal Validity: Subjects of these experiments are students but researchers have raised concerns over the use of students as subjects [33,34]. This will affect the generalization of the results. Also, the objects are programs prepared for and in academic environment.
- Construct Validity: The threats posed by conduct of the experiment was taken care of by the cross over design used. The design ensured that no subject worked on a program more than once nor did they use a technique more than once.
- Conclusion Validity: The results of the experiments were analyzed using non-parametric tests. Also, human judgement is involved in interpretation of defects detected. The subjects might not have applied the techniques in a very reliable way or report their findings correctly. Though, this was minimized because they expect to grades at the end of the experiments.

6. Conclusion

This paper attempted to aggregate the results of a series of experiment replications conducted to determine the effectiveness of three software evaluation techniques – code review by abstraction, decision coverage and equivalence partition as software evaluation techniques. Due to few numbers of replications available, the aggregation was done using a less rigorous aggregation technique recently proposed. The result of the technique revealed that overall, the technique, fault, program and their interactions are either significant or have a tendency to be; therefore, forcing a general rejection of the null hypothesis in their case. This implies performance of the technique is affected by the program type and fault type. Version that was introduced to increase the number of seeded faults was found not to be significant. A further study of the significant treatments and interactions revealed no difference in the performance of decision coverage and equivalence partition which are structural technique and functional technique respectively. Both performed better than code reading by stepwise abstraction.

References

[1] S. Vegas, Characterization schema for selecting software testing techniques, in: Facultad Informatica, Universidad Poilitecnica de Madrid, Spain, 2002.
[2] M.J. Harrold, Testing: a roadmap, in: ACM, 2000, pp. 61–72.
[3] N. Juristo, J. Morant, Common framework for the evaluation process of KBS and conventional software, Knowledge-Based Systems 11 (1998) 145–159.
[4] N. Ayewah, D. Hovemeyer, J.D. Morgenthaler, J. Penix, W. Pugh, Using static analysis to find bugs, Software, IEEE 25 (2008) 22–29.
[5] G. Tassey, The Economic Impacts of Inadequate Infrastructure for Software Testing, National Institute of Standards and Technology, RTI Project, 2002.
[6] T. Berling, T. Thelin, An industrial case study of the verification and validation activities, in: Software Metrics Symposium, 2003 Proceedings Ninth International, 2003, pp. 226–238.
[7] N. Juristo, A.M. Moreno, S. Vegas, Software Evaluation Techniques, Universidad Politecnica de Madrid, 2005. pp. 117.
[8] C. Catal, B. Diri, A systematic review of software fault prediction studies, Expert Systems with Applications 36 (2009) 7346–7354.
[9] R. Torkar, Towards Automated Software Testing, Techniques, Classifications and Frameworks, Classifications and Frameworks, Citeseer, 2006.
[10] C. Andersson, T. Thelin, P. Runeson, N. Dzamashvili, An experimental evaluation of inspection and testing for detection of design faults, in: Empirical Software Engineering, 2003, ISESE 2003, Proceedings, 2003 International Symposium on, 2003, pp. 174–184.
[11] S.S. So, S.D. Cha, T.J. Shimeall, Y.R. Kwon, An empirical evaluation of six methods to detect faults in software, Software Testing, Verification and Reliability 12 (2002) 155–171.
[12] D. Porto, M. Mendonca, S. Fabbri, CRISTA: a tool to support code comprehension based on visualization and reading technique, in: Program Comprehension, 2009, ICPC '09, IEEE 17th International Conference on, 2009, pp. 285–286.
[13] S.U. Farooq, S. Quadri, A Novel Approach for Evaluating Software Testing Techniques for Reliability, 2009.
[14] E. Fernández, Aggregation Process with Multiple Evidence Levels for Experimental Studies in Software Engineering, Citeseer, 2007. pp. 75–81.
[15] B. Olorisade, Summarizing the Results of a Series of Experiments: Application to the Effectiveness of Three Software Evaluation Techniques, LAP Lambert Academic Publishing, Germany, 2010.
[16] N. Juristo, A.M. Moreno, S. Vegas, Reviewing 25 years of testing technique experiments, Empirical Software Engineering 9 (2004) 7–44.
[17] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Transactions on Software Engineering 28 (2002) 721–734.
[18] N. Juristo, A.M. Moreno, Basics of Software Engineering Experimentation, Springer, 2001.
[19] N. Juristo, A. Moreno, S. Vegas, Limitations of empirical testing technique knowledge, Series on Software Engineering and Knowledge Engineering 12 (2003) 1–38.
[20] M. Ciolkowski, O. Laitenberger, S. Biffl, Software reviews, the state of the practice, Software, IEEE 20 (2003) 46–51.
[21] D.E. Perry, A.A. Porter, L.G. Votta, Empirical studies of software engineering: a roadmap, in: ACM, 2000, pp. 345–355.
[22] P. Runeson, C. Andersson, T. Thelin, A. Andrews, T. Berling, What do we know about defect detection methods? [software testing], Software, IEEE 23 (2006) 82–90.
[23] C. Wohlin, Experimentation in Software Engineering: An Introduction, Springer, 2000.
[24] V.R. Basili, R.W. Selby Jr., D.H. Hutchens, Experimentation in software engineering, in: DTIC Document, 1985.
[25] V.R. Basili, R.W. Selby, Comparing the effectiveness of software testing strategies, IEEE Transactions on Software Engineering (1987) 1278–1296.
[26] W.C. Hetzel, An experimental analysis of program verification, methods, 1976.
[27] G.J. Myers, A controlled experiment in program testing and code walkthroughs/inspections, Communications of the ACM 21 (1978) 760–768.
[28] E. Kamsties, C. Lott, An empirical evaluation of three defect-detection techniques, Software Engineering—ESEC'95 (1995) 362–383.
[29] M. Wood, M. Roper, A. Brooks, J. Miller, Comparing and combining software defect detection techniques: a replicated empirical study, Software Engineering—ESEC/FSE'97 (1997) 262–277.
[30] N. Juristo, S. Vegas, Functional testing, structural testing and code reading: what fault type do they each detect?, Empirical Methods and Studies in Software Engineering 2765 (2003) 208–232
[31] J.F. Hair, W.C. Black, B.J. Babin, R.E. Anderson, R.L. Tatham, Multivariate Data Analysis, sixth ed., Pearson Prentice Hall, NJ, USA, 2006.
[32] P. Runeson, A. Andrews, Detection or isolation of defects? An experimental comparison of unit testing and code inspection, in: 14th International Symposium on Software Reliability Engineering (ISSRE'03), IEEE, 2003, pp. 11.
[33] W.F. Tichy, Hints for reviewing empirical works in software engineering, Empirical Software Engineering: An International Journal 5 (2000) 309–312.
[34] M. Host, C. Wohlin, T. Thelin, Experimental context classification: incentives and experience of subjects, in: Software Engineering, 2005, ICSE 2005, Proceedings 27th International Conference on, 2005, pp. 470–478.
[35] B.K. Olorisade, Informal Aggregation Technique for Software Engineering Experiments, IJCSI International Journal of Computer Science Issues 9 (2012) 199–204.
[36] B. Beizer, Software Testing Techniques, second ed., International Thomson Computer Press, 1990.