e-Informatica Software Engineering Journal, Volume 11, Issue 1, 2017, pages: 77–102, DOI 10.5277/e-Inf170104

Efficiency of Software Testing Techniques: A Controlled Experiment Replication and Network Meta-analysis

Omar S. Gómez*, Karen Cortés-Verdín**, César J. Pardo***

*Facultad de Informática y Electrónica, Escuela Superior Politécnica de Chimborazo

**Facultad de Estadística e Informática, Universidad Veracruzana

***Electronic and Telecommunications Engineering Faculty, Information Technology Research Group (GTI), Universidad del Cauca

ogomez@espoch.edu.ec, kcortes@uv.mx, cpardo@unicauca.edu.co

Abstract

Background. Common approaches to software verification include static testing techniques, such as code reading, and dynamic testing techniques, such as black-box and white-box testing. **Objective.** With the aim of gaining a better understanding of software testing techniques, a controlled experiment replication and the synthesis of previous experiments which examine the efficiency of code reading, black-box and white-box testing techniques were conducted. **Method.** The replication reported here is composed of four experiments in which instrumented programs were used. Participants randomly applied one of the techniques to one of the instrumented programs. The outcomes were synthesized with seven experiments using the method of network meta-analysis (NMA). **Results.** No significant differences in the efficiency of the techniques were observed. However, it was discovered the instrumented programs had a significant effect on the efficiency. The NMA results suggest that the black-box and white-box techniques behave alike; and the efficiency of code reading seems to be sensitive to other factors. **Conclusion.** Taking into account these findings, the Authors suggest that prior to carrying out software verification activities, software engineers should have a clear understanding of the software product to be verified; they can apply either black-box or white-box testing techniques as they yield similar defect detection rates.

Keywords: software verification, software testing, controlled experiment, experiment replication, meta-analysis, network meta-analysis, quantitative synthesis

1. Introduction

Currently, due to the increase in both the size and complexity of software products, verification plays an important role in the software product development (or maintenance) process. The aim of software verification is to enssure that a software product fully satisfies all the requirements defined by the customer. It typically includes such activities as code executions, reviews, walkthroughs and inspections of the artifacts produced in the development or maintenance process.

Software verification is performed at different phases of the software development (or maintenance) process by following two approaches: reviewing or inspecting artifacts, such as documents and a source code (static approach) or an executing code (dynamic approach).

In the software construction phase, common techniques used in software verification include code reading (static approach), black-box and white-box testing (dynamic approach), and various other techniques, such as regression testing [1].

With the aim of gaining a better understanding of various software testing techniques applied during the software construction phase, in this work, the authors pursue two goals: 1) running a controlled experiment replication on the effi-

ciency of testing techniques expressed in terms of the number of defects detected per hour by each of the techniques: code reading, black-box and white-box (this carried out through the application of an experimental paradigm [2–4]), and 2) carrying out a synthesis of existing experiments which also address the efficiency of the related testing techniques.

Our replication is the extension of previous experiments reported in [5–12], where the effectiveness of the aforementioned software testing techniques was the main issue examined. In these experiments, the effectiveness was measured either as the percentage or as the number of defects observed in these testing techniques. Complementary to effectiveness, efficiency is another aspect that deserves attention. Due to the limitations of time and resources it is often raised in the software verification phase, it is worth considering which of the testing techniques behave in an optimal way (e.g. the fastest technique detecting defects). The authors have found some controlled experiments that also address the efficiency of the testing techniques [5–8, 11].

In order to corroborate the previous findings and also generate new knowledge with regard to the study of software testing techniques efficiency, this work reports the findings of a controlled experiment replication that examines efficiency in terms of the number of defects detected per hour of the following testing techniques: code reading, black-box and white-box testing. The replication results are then incorporated to existing related experiments following a quantitative synthesis approach. According to [13], this experiment can be considered as a conceptual replication of the original experiment reported in [5], only the constructs are maintained; these are the three testing techniques (causal constructs) and the efficiency (effect construct).

In science, replication is a key mechanism which allows for the verification of previous findings and for the consolidation of the body of knowledge [14, 15]. Replication is still a pending issue to be addressed in Software Engineering, since there is evidence showing a minimal amount of controlled experiments that have been replicated [16, 17]. If an experiment is not replicated or verified, there is no way to distinguish whether its outcome was produced by chance, artificially or it conforms to a reality. The results of this replication serve as a mechanism for verification, and they also contribute to the consolidation of the body of knowledge in the software verification research area. Although a number of experiments related to our replication have been conducted, it is worth to note that increasing the number of related experiments (experiments family) will allow other researchers to apply quantitative synthesis methods in a more confident way, the synthesis outcome will be strengthened by the pooled samples sizes of the related experiments.

The rest of the document is organized as follows. In Section 2, the related work is presented. In Section 3, the baseline experiment of the presented replication is described. In Section 4, the studied software testing techniques are studied. Sections 5 presents the context of our experiment replication. In Section 6,there is the statistics used for analysis and the results obtained. In Section 7, a quantitative synthesis using the obtained results and the results from related experiments is carried out. In Section 8, the findings are discussed and finally in Section 9 the conclusions are presented.

2. Related work

This section presents the summary of the empirical studies (family or series of experiments) related to the experiment replication reported here. The authors considered the controlled experiment reported in [5] as the baseline for their experiments. The aim of this experiment is to examine the effectiveness, efficiency and cost of three software testing techniques: black-box by equivalence class partitioning and boundary value analysis, white-box by sentence coverage and code reading by stepwise abstraction.

The authors of [5] carried out two replications of their experiment. Years later, the authors of [6, 7] performed the other two replications. A few years later, the authors in [8] conducted another replication. The authors of [9, 10] also carried out several replications. Recently, the au-

thors of [11] and [12] replicated the experiment as well.

Note that these replications do not take as reference the same baseline experiment. For example, the second and third replication reported in [5] takes as baseline their first experiment. The experiment reported in [6, 7] is the replication of [5]. In the case of the experiment reported in [8], the authors used the experiment replication package of [6, 7], thus considering this experiment the replication of [6, 7]. Experiments of the authors of [9, 10] are based on the replication packages of [6, 7] and [8]. With regard to the experiment in [11], it is related to the replication package of [6, 7]. In the case of the experiment reported in [12], the authors adapted the replication package of [9,10]. Table 1 presents some characteristics of these experiments.

2.1. Constructs and operationalizations studied

2.1.1. Cause constructs and operationalizations

The cause constructs examined in these experiments are: the black-box [5–12], white-box [5–12] and code reading [5–11] techniques. Regarding black-box, it was operationalized either as equivalence class partitioning and boundary value analysis [5–8, 11] or as equivalence class partitioning [9, 10, 12]. Concerning white-box, it was operationalized either as sentence coverage [5] or as branch coverage [6–12]. In the case of code reading, in all the experiments [5–12] it was operationalized by the use of stepwise abstractions approach [18]. Secondary cause constructs, also examined, are the instrumented program (software type) [5–12], the participant expertise [5], the defect type [9, 10] and the version of the instrumented programs [9, 10].

2.1.2. Effect constructs and operationalizations

The effect constructs examined in these experiments are: effectivenesses [5–12], efficiency [5–8, 11], fault visibility [9] and cost [5–7, 11].

The **effectiveness** construct was operationalized as the number of observed defects [5, 8], the percentage of observed defects [5–7, 11, 12], the number of observable defects [5], the percentage of observable defects [5, 12], the percentage of participants who detect a given defect for each defect in the instrumented program [9, 10], the percentage of participants that are able to generate a test case that uncovers the failure associated with a given defect [9,10], the number of isolated defects [8], and the percentage of isolated defects [6, 7, 11]. **Efficiency** was operationalized as the number of defects detected per hour (detection rate) [5–8, 11], and as the number of defects isolated per hour (isolation rate) [6, 7]. Finally **cost** was operationalized as the time spent applying the testing techniques [5–7, 11], defect isolation time [6, 7, 11], cpu-time [5], connect time [5] and number of programs runs [5].

2.2. Findings

In this section some relevant findings of these experiments are presented. These findings are organized according to the different effect constructs examined.

2.2.1. Effectiveness

Number of observed defects (operationalization o1.1). For the umd82 experiment [5], either code reading or black-box were significantly more effective than white-box. Concerning umd83 experiment [5], no significant differences were observed between the three testing techniques. In the case of umd84 [5], code reading was significantly more effective than black-box and white-box, also black-box was significantly more effective than white-box. In the case of uos97 [8], the authors observed a significant difference in the effectiveness of the techniques, however, it is not described which of the pairwise techniques was significantly different. It seems that black-box and white-box behave in a similar way and that these techniques are more effective than code reading. With regard to the studied secondary factors and interaction effects:

– Software type (instrumented programs). The effectiveness of the techniques (measured as the number of observed defects) was signifi-

Table 1. Characteristics of the aforementioned family of experiments

Experiment	Participants	Programs and number of defects	Language	Country
umd82 [5]	CS (under)graduates	p1(9), p2(6), p3(7)	Simplt	USA
umd83 [5]	CS (under)graduates	p1(9), p2(6), p4(12)	Simplt	USA
umd84 [5]	Professionals	p1(9), p3(7), p4(12)	Fortran	USA
ukl94 [6, 7]	CS undergraduates	nt(11),cm(14),na(11)	C	Germany
ukl95 [6, 7]	CS undergraduates	nt(6), cm(9), na(7)	C	Germany
uos97 [8]	CS undergraduates	nt(8), cm(9), na(8)	C	UK
upm00 [9]	CS undergraduates	nt(9), cm(9), na(9), tr(9)	C	Spain
upm01 [9, 10]	CS undergraduates	nt(7), cm(7), na(7)	C	Spain
upm02 [10]	CS undergraduates	nt(7), cm(7), na(7)	C	Spain
upm03 [10]	CS undergraduates	nt(7), cm(7), na(7)	C	Spain
upm04 [10]	CS undergraduates	nt(7), cm(7), na(7)	C	Spain
upm05 [10]	CS undergraduates	nt(7), cm(7), na(7)	C	Spain
uds05 [10]	CS undergraduates	nt(7), cm(7), na(7)	C	Spain
upv05 [10]	CS undergraduates	nt(7), cm(7), na(7)	C	Spain
ort05 [10]	CS undergraduates	cm(7), na(7)	C	Uruguay
uok11 [11]	CS graduates	nt(8), cm(9), na(8)	C	India
uady13 [12]	CS undergraduates	nt(7), cm(7)	C	Mexico

cantly affected by the instrumented programs used in umd84 [5] and uos97 [8]. On the other hand, the effectiveness is not affected by software type in umd82 and umd83 [5].

– Expertise. The effectiveness (in terms of the number of observed defects) was significantly affected by the expertise, advanced expertise participants detected more defects than either intermediates or juniors (umd84 [5]).
– Interaction effects. In umd83 [5] and uos97 [8], the authors report a significant interaction effect between the testing techniques and the instrumented programs. A three-way interaction between techniques, programs and expertise was observed in umd84 [5].

Percentage of observed defects (o1.2). Either code reading or black-box were significantly more effective than white-box (in umd82 [5]). Code reading was significantly more effective than black-box and white-box, and also black-box was significantly more effective than white-box (in umd84 [5]). There are no significant differences between the testing techniques (umd83 [5], ukl94, ukl95 [6, 7], uok11 [11] and uady13 [12]). In the case of secondary factors and interaction effects:

– Software type. The effectiveness of the techniques (measured as the percentage of observed defects) was significantly affected by the instrumented programs in umd82, umd83, umd84 [5], ukl94 [6, 7] and uok11 [11].
– Expertise. The effectiveness significantly varies with regard to the level of expertise (in umd84 [5]). The percentage of observed defects was significantly higher for the participants with advanced expertise, this difference is significant only with respect to juniors. There were not significant differences between intermediates and juniors in umd82, umd83, umd84 [5].
– Interaction effects. In umd83 [5] an interaction effect between the testing techniques and the instrumented programs was observed. A three-way interaction between techniques, programs and expertise was observed in umd84 [5].

Number of observable defects (o1.3). In the case of umd82 [5], the number of observable defects was significantly higher for black-box (in comparison to white-box). Significant differences were not found in umd84 [5].

Percentage of observable defects (o1.4). The percentage of observable defects is significantly higher for black-box than for white-box in umd82 [5]. Significant differences were not found in umd84 [5] and uady13 [12].

Percentage of participants who detect a given defect for each defect in the instru-

mented program (o1.5). The effectiveness is affected by the testing techniques. Code reading is significantly less effective than black-box and white-box, and black-box and white-box behave in a similar way (upm00 [9], upm01, upm02, upm03, upm04, upm05 and uds05 [10]). Concerning secondary factors and interaction effects:

- Software type. The effectiveness of the techniques was significantly affected by the instrumented programs in upm00 [9].
- Defect type. In upm00 [9], the effectiveness of the techniques was significantly affected by the defect types injected in the instrumented programs.
- Interaction effects. In upm00 [9] an interaction effect between the testing techniques and the instrumented programs was observed. Also an interaction effect between the instrumented programs and the defect types was observed.

Percentage of participants that are able to generate a test case that uncovers the failure associated with a given defect (o1.6). The effectiveness did not impact black-box and white-box (upm01, upm02, upm03, upm04, upm05, uds05 and upv05 [9, 10]). Black-box is significantly more effective than white-box (ort05 [10]). In the case of secondary factors and interaction effects:

- Software type. The effectiveness of the techniques was significantly affected by the instrumented programs in upm01, upm05, uds05, upv05 and ort05 [10].
- Defect type. In upm04, uds05, upv05 and ort05 [10], the effectiveness of the techniques was significantly affected by the defect types injected in the instrumented programs.
- Program version. The version of the instrumented programs was not affected in upm01, upm02, upm03, upm04, upm05, uds05, upv05 and ort05 [10].
- Interaction effects. In upm00, upm01, upm02, upm03, upm04, upm05, upv05 and ort05 [9, 10] an interaction effect between the testing techniques and the instrumented programs was observed. An interaction effect between the instrumented programs and the defect types was observed in upm00, upm02, upm04, upm05, uds05, upv05 and ort05 [9,10]. An interaction effect between techniques and defect types was observed in upm01, upm03, upm05 and uds05 [10]. An interaction effect between program version and defect types was observed in upm03, upv05 [10]. Another interaction effect between the technique and the program version was observed in uds05 and upv05 [10]. Three-way interactions between instrumented programs, techniques and defect types, and also between instrumented programs, program versions and defect types were observed in ort05 [10].

Number of isolated defects (o1.7). Although some information about this is presented in uos97 [8] neither descriptive nor inferential analysis is discussed.

Percentage of isolated defects (o1.8). The effectiveness of the testing techniques behaves in a similar way (ukl94 [6, 7] and uok11 [11]). The percentage of isolated defects is significantly affected by the testing techniques in ukl95 [6, 7], although a post-hoc is missing, it seems that black-box and code reading show better effectiveness than white-box. The findings for secondary factors and interaction effects are:

- Software type. The effectiveness of the techniques was significantly affected by the instrumented programs (in ukl94 [6, 7] and uok11 [11]).
- Technique application order (sequence). The effectiveness of the techniques is significantly affected by the order in which techniques are applied (in ukl94 [6, 7]).

Summarizing. It can be observed that the effectiveness construct has the greatest number of operationalizations. It was operationalized in several ways. It can also be seen that secondary factors such as instrumented programs and expertise may have an impact on the techniques effectiveness. It is not so clear which of the techniques is more effective due to contradictory findings.

2.2.2. Efficiency

Defects detected per hour (o2.1). The three testing techniques showed similar defect detection rates in umd82, umd83 [5] and uok11 [11].

Code reading showed the higher defect detection rate in comparison to either black-box or white-box (umd84 [5]), this difference was significant. The authors of ukl94 and ukl95 experiments [6,7] report a significant difference between the techniques, however, a post-hoc analysis did not show the pairwise significant differences, it seems that black-box shows the higher defect detection rate. In the case of uos97 [8], the authors did not report the inferential statistics for this metric, however, black-box seems to yield the higher defect detection rate, white-box appears to be the second most efficient technique. The findings from secondary factors and interaction effects are:

- Software type. The efficiency of the techniques (measured as the number of defects detected per hour) was significantly affected by the instrumented programs in umd82, umd84 [5] and uok11 [11].
- Expertise. The efficiency did not vary with regard to the level of expertise (umd83, umd84 [5]). Intermediate participants detected defects at a significantly faster rate than juniors did (umd82 [5]).
- Technique application order (sequence). The efficiency of the techniques is significantly affected by the order in which they are applied (in ukl95 [6, 7]).
- Interaction effects. A two-way interaction between techniques and instrumented programs was observed in umd84 [5].

Defects isolated per hour (o2.2). The three techniques behave in a similar way (ukl94 [6, 7]). However, in the case of ukl95 [6,7] and uok11 [11], the defect isolation rate is significantly affected by the techniques, although a post-hoc analysis is missing, in ukl95 [6, 7] it seems that black-box shows a higher defect isolation rate. In the case of uok11 [11] it seems that white-box and black-box show higher defect isolation rates than code reading. With regard to secondary factors:

- Technique application order (sequence). The defect isolation rate is significantly affected by the order in which techniques are applied (ukl94 [6, 7]).

Summarizing. Similar findings can be observed for the efficiency construct, secondary factors, such as instrumented programs, expertise and the technique application order, may have an impact on the techniques efficiency. At first sight, it is hard to conclude which of the techniques is more efficient due to some contradictory findings.

2.2.3. Cost

Time spent applying the testing techniques (o3.1). The time spent applying the three testing techniques is similar (in umd83, umd84 [5] and uok11 [11]). Applying white-box requires significantly more time than applying either code reading or black-box (umd82 [5]). Although a significant difference was observed in ukl94 and ukl95 [6,7], the authors did not present a post-hoc analysis to assess which of the techniques requires significantly less time, however, it seems that applying code-reading requires more time than applying white-box; black-box requires less time than white-box (ukl94, ukl95 [6, 7]). In the case of secondary factors and interaction effects:

- Software type. The time spent applying the techniques was significantly affected by the instrumented programs in umd82, umd84 [5] and uok11 [11].
- Expertise. The time spent applying the techniques did not vary with regard to the level of expertise (umd82, umd83, umd84 [5]).
- Technique application order (sequence). The time spent applying the techniques is significantly affected by the order in which they are applied (in ukl95 [6, 7]).
- Interaction effects. A two-way interaction between techniques and instrumented programs was observed in umd84 [5].

Defect isolation time (o3.2). The experiments in ukl94, ukl95 [6, 7] and uok11 [11] report a significant difference between the techniques, however, a post-hoc analysis does not identify pairwise significant differences. Code reading seems to require less time for isolating defects than the other techniques.

Cpu-time (o3.3). Black-box required significantly more cpu-time than white-box (in umd84 [5]).

Connect time (o3.4). Participants applying black-box black-box spent significantly more minutes of connect time than those applying white-box (in umd84 [5]).

Number of program runs (o3.5). This metric did not show significant differences between black-box and white-box (in umd84 [5]).

Summarizing. Secondary factors, such as instrumented programs, expertise and the technique application order, may have an impact on the cost of applying the testing techniques. With regard to the the application time of these techniques, it is hard to identify which of the testing techniques incurs fewer costs. However, code reading seems to require less time for isolating defects. Concerning cpu-time and connect time, black-box seems to demand more resources.

To conclude this section, Table 2 shows the global summary of the findings found in this family of experiments.

3. Baseline experiment

Following the proposed guidelines for reporting experiment replications [19], this section describes the original experiment. In [5], the authors report results from three controlled experiments which were conducted as controlled experiments where different types of participants (undergraduate, graduate students and practitioners) applied three software testing techniques (code reading, black-box testing and white-box testing) to four instrumented programs.

The participants in these experiments were representative of three levels of computer science expertise: junior (0–2 years of experience), intermediate (2.5–6.2 years of experience) and advanced (10 years of experience). A total of 29, 13 and 32 people participated in three respective experiments. In the first two experiments, the participants were either upper-level computer science majors or graduate students. In the third experiment, the participants were programming professionals from NASA and the Computer Sciences Corporation.

The instrumented programs used in these experiments were coded in Fortran and Simpl-T. The four programs are related to a text processor (p1), a mathematical plotting routine (p2), a numeric abstract data type (p3) and a database maintainer program (p4). Table 3 shows some characteristics of the used programs, such as source lines of code (SLOC), cyclomatic complexity (VG) and the number of defects injected.

It is worth noting that the authors did not use all the programs in the three experiments. Programs p1, p2 and p3 were used in the first experiment; programs p1, p2 and p4 were used in the second experiment, and programs p1, p3, and p4 were used in the third one.

The testing techniques examined in [5] were code reading by stepwise abstraction [18], black-box testing through equivalence partitioning and boundary value analysis [20, 21] and white-box testing through statement coverage [21, 22]. Table 4 shows the efficiency observed (in terms of defects detected per hour) in the experiments and their standard deviations. The authors only report a significant difference (at $\alpha < 0.0003$) in the third experiment. This difference shows an enhanced efficiency for the code reading technique.

Regarding the defect detection rates in the instrumented programs used in the experiments, Table 5 shows the defect detection rates per program and their standard deviations. The authors report a significant difference in the first (at $\alpha < 0.01$) and third experiment (at $\alpha < 0.0001$). In both experiments, the testing techniques showed higher levels of efficiency in program p3 (Data type).

The authors also examined the efficiency of the participants according to their differing levels of expertise: junior, intermediate and advanced. Table 6 shows the efficiency rates of these types of participants and their standard deviations.

The athors report a significant difference only in the first experiment. Intermediate participants detected defects at a faster rate than junior participants. In the remaining experiments, the authors did not observe any significant difference in defect detection rates between expertise levels.

Table 2. Global summary of findings of this family

EFFECTIVENESS

	umd82	umd83	umd84	ukl94	ukl95	uos97	upm00	upm01	upm02	upm03	upm04	upm05	uds05	upv05	ort05	uok11	uady13
o1.1	(✠ ≡ ■) > □	✠ ≡ ■ ≡ □	✠ > ■ > □ SF:sw,exp I:sw,exp			(■ ≡ □) > ✠ SF:sw I:sw											
o1.2	(✠ ≡ ■) > □ SF:sw	✠ ≡ ■ ≡ □ SF:sw I:sw	✠ > ■ > □ SF:sw,exp I:sw, exp	✠ ≡ ■ ≡ □ SF:sw	✠ ≡ ■ ≡ □											✠ ≡ ■ ≡ □ SF:sw	✠ ≡ ■ ≡ □
o1.3	■ > □		■ ≡ □														
o1.4	■ > □		■ ≡ □														■ ≡ □
o1.5							(■ ≡ □) > ✠ SF:sw,dt I:sw,dt	(■ ≡ □) > ✠	(■ ≡ □) > ✠	(■ ≡ □) > ✠	(■ ≡ □) > ✠	(■ ≡ □) > ✠					
o1.6								■ ≡ □ SF:sw I:sw	■ ≡ □ I:sw,dt	■ ≡ □ I:sw,v	■ ≡ □ SF:dt I:sw,dt	■ ≡ □ SF:sw I:sw,dt	■ ≡ □ SF:sw,dt I:sw,dt,v	■ ≡ □ SF:sw,dt I:sw,dt,v	■ > □ SF:sw,dt I:sw,dt,v		
o1.8				✠ ≡ ■ ≡ □ SF:sw I:seq	(✠ ≡ ■) > □ SF:sw											✠ ≡ ■ ≡ □ SF:sw	

EFFICIENCY

	umd82	umd83	umd84	ukl94	ukl95	uos97	upm00	upm01	upm02	upm03	upm04	upm05	uds05	upv05	ort05	uok11	uady13
o2.1	✠ ≡ ■ ≡ □ SF:sw,exp	✠ ≡ ■ ≡ □	✠ > (■ ≡ □) SF:sw I:sw	✠ ≡ ■ ≡ □	✠ ≡ ■ ≡ □ SF:seq	■ > □ > ✠										✠ ≡ ■ ≡ □ SF:sw	
o2.2				✠ ≡ ■ ≡ □ SF:seq	■ > ✠ > □											(■ ≡ □) > ✠	

COST

	umd82	umd83	umd84	ukl94	ukl95	uos97	upm00	upm01	upm02	upm03	upm04	upm05	uds05	upv05	ort05	uok11	uady13
o3.1	□ > (✠ ≡ ■) SF:sw	✠ ≡ ■ ≡ □	✠ ≡ ■ ≡ □ SF:sw I:sw	✠ > □ > ■	SF:seq											✠ ≡ ■ ≡ □ SF:sw	
o3.2				■ > □ > ✠	■ > □ > ✠											■ > □ > ✠	
o3.3			■ > □														
o3.4			■ > □														
o3.5			■ ≡ □														

Effectiveness operationalizations. Number of observed defects **(o1.1)**; Percentage of observed defects **(o1.2)**; Number of observable defects **(o1.3)**; Percentage of observable defects **(o1.4)**; Percentage of participants who detect a given defect for each defect in the instrumented program **(o1.5)**; Percentage of participants that are able to generate a test case that uncovers the failure associated with a given defect **(o1.6)**; Percentage of isolated defects **(o1.8)**.
Efficiency operationalizations. Defects detected per hour **(o2.1)**; Defects isolated per hour **(o2.2)**.
Cost operationalizations. Time spent applying the testing technique **(o3.1)**; Defect isolation time **(o3.2)**; Cpu-time **(o3.3)**; Connect time **(o3.4)**; Number of programs runs **(o3.5)**.
Software testing techniques. Code reading **(✠)**; Black-box **(■)**; White-box **(□)**.
Testing techniques comparisons convention. The technique behaves significantly better **(>)**; The technique behaves in a similar way **(≡)**, non-significant difference.
Secondary factors (SF) studied. These secondary factors showed a significant difference in the ANOVA: Instrumented software program **(sw)**; Instrumented software program version **(v)**; Defect type **(dt)**; Programmer expertise **(exp)**; Technique application order **(seq)**.
Interactions (I). In this case some secondary factors were involved in a significant interaction with the main factor (software testing technique), or interactions were found between some secondary factors.

Table 3. Characteristics of instrumented programs used in [5]

Program	SLOC	VG	Defects
Formatter (p1)	169	18	9
Plotter (p2)	145	32	6
Data type (p3)	147	18	7
Database (p4)	355	57	12

Table 4. Average and standard deviation of defect detection rates per software testing technique

Technique	umd82 [5]	umd83 [5]	umd84 [5]
Code reading	1.90 (1.83)	0.56 (0.46)	3.33 (3.42)
Black-box	1.58 (0.90)	1.22 (0.91)	1.84 (1.06)
White-box	1.40 (0.87)	1.18 (0.84)	1.82 (1.24)

Table 5. Average and standard deviation of defect detection rates per software program

Program	umd82 [5]	umd83 [5]	umd84 [5]
Formatter (p1)	1.60 (1.39)	0.98 (0.67)	2.15 (1.10)
Plotter (p2)	1.19 (0.83)	0.92 (0.71)	–
Data type (p3)	2.09 (1.42)	–	3.70 (3.26)
Database (p4)	–	1.05 (1.04)	1.14 (0.79)

Table 6. Average and standard deviation of defect detection rates according to level of expertise

Expertise	umd82 [5]	umd83 [5]	umd84 [5]
Junior	1.36 (0.97)	1.00 (0.85)	2.14 (2.48)
Intermediate	2.22 (1.66)	0.96 (0.74)	2.53 (2.48)
Advanced	–	–	2.36 (1.61)

4. Description of the studied software testing techniques

The following subsections summarize the software testing techniques known as code reading, black-box and white-box testing which were used in this experiment replication.

4.1. Code reading

The aim of code reading is to find defects in code documents without executing the code or the software (static approach).

The studied code reading technique is known as stepwise abstraction [18]. In code reading by stepwise abstraction, a software engineer identifies methods (or functions) in the source code, and then he or she abstracts from them the software program functionality. A set of abstractions builds up to other abstractions which represent modules and so forth. This process is followed until a conceptual understanding of the prime abstraction emerges and brings into view an overall picture of the examined code. This abstraction is then compared to the product specification with the aim of finding inconsistencies or defects in the source code.

4.2. Black-box testing

This type of software testing technique is based on the software product specification. Once a software engineer has the specification, he or she starts to design a set of test cases. The software to be verified is seen as a black-box whose behavior is only determined by studying its inputs and examining its outputs. Nevertheless, because examining all the possible inputs is impractical, only a subset of inputs is selected for testing during the software product verification.

The software engineer assumes that the software product to be verified contains a set of inputs that will probably cause the product to fail. As a consequence of introducing these inputs, the product yields outputs which reveals the presence of defects. Because exhaustive testing is impractical, the main goal is to find a set of data inputs whose probability of belonging to the set of in-

puts that produce a failure in the product is as high as possible [21, 23]. There are strategies for designing test cases to reveal these inputs. Two such strategies are known as: equivalence class partitioning (ECP) and boundary-value analysis (BVA). The authors worked with the ECP approach, where an equivalence class represents a set of valid or invalid states that are defined as input conditions. A typical input condition is a specified numerical value, a range of values, a set of related values (such as categories) or a logical condition.

4.3. White-box testing

It is also known as crystal or transparent testing, the aim of this technique is to design test cases that are able to exhaustively cover the software code, examining all aspects of the structure and logic of the software product. The main idea is to design test cases that execute all code sentences at least once and that also execute all branches of code containing conditions (evaluating both branches by using both true and false expressions) [21,23]. Because examining all paths of the software code can be impractical, various strategies exist for achieving adequate code coverage. Some of these strategies include: statement coverage, decision (or branch) coverage and condition coverage. The authors worked with the branch coverage approach where a set of test cases is designed to ensure that each control structure is executed at least once. To assess this technique, the programs with the Java JCov coverage, a tool which provides a means to measure and analyze dynamic code coverage of Java programs, were instrumented.

5. Experiment replication context

The experiment replication reported here is composed of four comparative studies (controlled experiments) carried out in December 2014 at the Technical School of Chimborazo (ESPOCH) as part of a software verification workshop. The participants were undergraduate students in their last semester of the software systems engineering bachelor degree. According to [24], the participants were categorized as advanced beginners, i.e. students having a working knowledge of the key aspects of software development practice.

The workshop was offered at no cost and it was intended for students in their last semester so as to complement their technical skills with a software verification course. Since the workshop was voluntary and free of charge, coercion was avoided. The participants were told that they could leave the workshop at any moment. Verbal consent was given from all the participants; the main goals of the experiment were explained to the participants and they were told that the experiment was part of a software verification workshop.

A differently instrumented software program was used in each experiment. The program sizes ranged between 253 and 392 SLOC. Programs were coded in the Java programming language. The average cyclomatic complexity (VG) of programs was around 40. Each program had the same type and number of defects injected (6 defects). As reference, the defect classification scheme of [25] was used, it is the same scheme as the one used in the baseline experiment [5] and also in the family discussed in Section 2. However, regarding one of the defect classification schemes, only three defect types (cosmetic, initialization and control) were used instead of the six used in [5] (cosmetic, initialization, control, data, interface and computation). The change was made to have better contol over experimental conditions, and thus havie the same number and defect types. The defects injected in each instrumented program were as follows:

- omission – cosmetic (F1),
- omission – initialization (F2),
- omission – control (F3),
- commission – cosmetic (F4),
- commission – initialization (F5),
- commission – control (F6).

It can be seen that all defect types were equally balanced in each software program, thus there was more experimental control over the instrumented programs.

The same defect counting scheme as the one used here was also applied in [6, 7], a failure is

Table 7. Characteristics of instrumented programs used in the study

Program	SLOC	VG	Defects	Session type	n
Triangle	41	1	1	Training	16
Deviation	184	14	3	Training	15
Banking	253	28	6	Experiment 1	15
Nametbl	392	43	6	Experiment 2	13
Ntree	349	46	6	Experiment 3	13
Cmdline	300	45	6	Experiment 4	12

observed if the participant applying one of the techniques records the deviate behaviour of the instrumented program with regard to its specification. In code reading, an inconsistency (analog to a failure) is observed if the participant records the inconsistency between his or her abstractions and the specification. False positives which are perceived defects reported by participants that are not in fact defects were ignored.

The experiments were run as part of a software verification workshop. This workshop consisted of ten sessions conducted on alternate days, where each session lasted between two and three hours. The first sessions were used to teach the use of the software testing techniques. Two sessions were used for training, where the participants applied the testing techniques to two instrumented programs. Table 7 shows the used program characteristics, the session type and the number of participants per session.

Regarding program functionality, the Triangle software program determines the type of triangle defined given three input values. Deviation calculates the average and standard deviation of n numbers. The banking program implements basic functions for managing bank accounts. Nametbl implements basic functions for managing a table of symbols. Ntree implements functions for managing an N-ary tree. Finally, cmdline implements the basic functionality of a command line program. All the programs were developed and instrumented by a student enrolled in his last year of the software engineering bachelor degree, he was under our supervision during a semester. The following programs were used as reference: nametbl, ntree and cmdline used in [10], these three programs were entirely rewritten to the Java programming language and instrumented with the previously mentioned defects.

5.1. Experiment replication goal

Following the GQM approach [26] this controlled experiment replication was defined as: "Analyze the testing techniques black-box, white-box and code reading for the purpose of comparison with regard to their efficiency (defects detected per hour) from the point of view of the researcher in an academic controlled context using small instrumented Java programs."

5.2. Research questions

For this controlled experiment replication, the following main research questions were stated:

- **RQ1**. Is efficiency affected by the studied testing techniques?
- **RQ2**. Do instrumented software programs impact the efficiency of the software testing techniques?
- **RQ3**. Does the relationship between techniques and programs affect the efficiency?

With the collected data of this experiment replication it is possible to define a secondary research question linked to a secondary analysis (defect analysis). This secondary question seeks to explore a possible impact on the software testing techniques efficiency and the defect classification schemes used in the instrumented programs. This secondary research question (SRQ1) was defined as follows:

- **SRQ1**. Do defect types (according to used defect classification schemes) impact the efficiency of the studied software testing techniques?

Table 8. Factorial design structure used

Technique/ Program	Exp. 1 Banking (ba)	Exp. 2 Nametbl (na)	Exp. 3 Ntree (nt)	Exp. 4 Cmdline (cm)
Code reading (cr)	cr, ba	cr, na	cr, nt	cr, cm
Black-box (bb)	bb, ba	bb, na	bb, nt	bb, cm
White-box (wb)	wb, ba	wb, na	wb, nt	wb, cm

The efficiency construct is operationalized according to the number of defects detected per hour after applying the testing techniques. To answer the previous research questions, three hypotheses were defined. For RQ1, the null hypothesis is defined as follows: All the testing techniques studied have similar or equal levels of efficiency. For RQ2, the null hypothesis to test is as follows: The type of software program does not affect the efficiency of testing techniques. For RQ3, the null hypothesis is defined as follows: Efficiency is not affected by the relationship between testing techniques and the type of software program. With regard SRQ1 the null hypothesis is defined as: the defect classification schemes used in the instrumented programs do not affect the efficiency of the testing techniques.

5.3. Design and execution

The four experiments constitute a factorial design (3×4) with two factors (technique and program), where the factor technique is composed of three levels (code reading, black-box and white-box testing) and the factor program is composed of four levels (banking, nametbl, ntree and cmdline programs). A factorial design allows for the study of several factors and the interactions among them. The factorial design layout for this replication is shown in Table 8. A completely randomized design was used in each experiment. At the beginning of each session, treatments (techniques) were randomly assigned to participants. In each session, every participant applied a testing technique to an instrumented software program.

The experiments were conducted in December 2014 as part of a workshop on software verification at ESPOCH. Participants used a web application for registering information regarding the application of the software testing technique to a given instrumented program. In a non-invasive way, this web application collected the time that participants spent performing the testing techniques. Below, we provide an overview of how each testing technique is applied on an instrumented program during the training and experiment sessions.

Code reading. Participants used code reading by stepwise abstraction [18]. Each participant receives the source code of the software. Then the participant inspects the code and starts to generate abstractions in a natural language. After the participant has constructed the prime abstraction, he or she is provided with the product specification. Then the participant compares his or her abstractions with the product specification and any inconsistencies observed are registered as defects. The time elapsed for carrying out the previous activities is taken into account for computing the number of defects detected per hour.

Black-box. Participants followed the equivalence class partitioning approach. Each participant receives the software product specification and then begins to generate valid and invalid equivalence classes. Next, the participant designs test cases from the equivalence classes defined and registers the expected outputs. The participant then executes the test cases by running the software program and registers the observed output from each test case. The participant then compares the expected outputs to the observed outputs, and any inconsistencies are registered as defects.

White-box. Participants receive the source code and the instrumented software program. Each participant then starts to generate test cases with the aim of achieving 100% branch coverage of the source code. The participant then registers the observed outputs after running the program. For this testing technique, software programs were instrumented with the Java JCov coverage

Table 9. The collected defect detection rate measurements

Technique/ Program	Exp. 1 Banking (ba)	Exp. 2 Nametbl (na)	Exp. 3 Ntree (nt)	Exp. 4 Cmdline (cm)
Code reading (cr)	0, 1.56, 0.45	1.58, 0.67	0, 1.07	0, 0.71
	0, 0	0, 0	1.86, 0.6	0, 0.72
Black-box (bb)	0.55, 2.26, 0.91	0.89, 0	0.43, 1.12, 0	0, 0.5
	1.09, 1.07	0, 0	0.44, 0	0, 0.47
White-box (wb)	1.1, 0.5, 0	1.28, 1.38, 0	1.03, 0	0.47, 0
	1.1, 4.44	1.04, 0.26	0.52, 0	0, 0

tool, so the participants using this technique were able to see the percentage of coverage achieved after each test case execution. Once a participant achieves the maximum coverage level, he or she the gains access to the product specification. Next, the participant registers the expected outputs as defined by the product specification. He or she then compares the observed outputs with the expected outputs, and any inconsistencies are registered as defects.

In the case of the two dynamic techniques (black-box and white-box), the time elapsed for generating and running the test cases (which encompasses the activities previously mentioned) is taken into account for computing the number of defects detected per hour.

With the aim of striving towards better research practices in SE [27] all the collected measurements are reported. These raw data will help other researchers to verify or re-analyze [28] the experiment results presented in this work. Table 9 shows all the efficiency measurements (defect detection rates) collected during the experiment sessions (the raw data is available in Appendix). A total of 53 measurements were collected, this sample size is slightly greater than the average sample size used in software engineering experiments [16].

6. Analysis and results

This section presents both the collected descriptive and inferential statistics for the efficiency measurements. Table 10 shows the mean defect detection rates and their standard deviations for the testing techniques assessed in the four experiments.

As shown in Table 10, there is not a clear distinction between the efficiency of the different testing techniques. In the first and second experiments, white-box testing seems to be more efficient than black-box testing and code reading, however, in the third and fourth experiments, code reading performs better. With respect to the instrumented programs, Table 11 shows the mean defect detection rates and their standard deviations for the instrumented programs used in the four experiments.

As shown in Table 11, efficiency seems to vary depending on the program. The software program identified as banking, on average, yields an efficiency rate of 1 defect per hour. This program has the data point with the maximum efficiency rate. Conversely, cmdline shows the worst efficiency rate; on average, the efficiency in this program yielded 0.24 defects detected per hour.

Descriptive statistics give us an overview of basic features of the collected efficiency measurements, but at this point, it is not possible to draw any confident conclusions with respect to possible differences between treatments. Once the overview of the data is provided, it is possible to continue testing the hypotheses previously stated using inferential statistics The four experiments can be arranged in a factorial experiment design [3]. The statistical model employed according to the factorial design (3×4) is defined in Equation (1).

$$y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \epsilon_{ijk}\,. \qquad (1)$$

In this equation μ is the grand mean, α_i represents the effect of software testing technique i, β_j represents the effect of program j, $(\alpha\beta)_{ij}$ is the interaction effect between treatments i

Table 10. Average defect detection rates and standard deviations of the software testing techniques

Technique	Exp. 1	Exp. 2	Exp. 3	Exp. 4
Code reading	0.4 (0.68)	0.56 (0.75)	0.88 (0.79)	0.36 (0.41)
Black-box	1.18 (0.64)	0.22 (0.44)	0.4 (0.46)	0.24 (0.28)
White-box	1.43 (1.75)	0.79 (0.62)	0.39 (0.49)	0.12 (0.24)

Table 11. Average defect detection rates and standard deviations per instrumented program

Program	Exp. 1	Exp. 2	Exp. 3	Exp. 4
Banking	1 (1.15)	–	–	–
Nametbl	–	0.55 (0.62)	–	–
Ntree	–	–	0.54 (0.58)	–
Cmdline	–	–	–	0.24 (0.31)

and j, k is the number of replications in each treatment combination, and ϵ is the random error which assumes $N(0, \sigma^2)$. The analysis of variance (ANOVA) [2–4] is used to assess the components of the model (such as technique, program and the interaction between technique and program).

Before drawing any conclusions related to the components of the model, it is necessary to assess: 1) that the collected measurements are independent (independence), 2) that the variance is the same for all the measurements (homogeneity), and 3) that the measurements follow a normal distribution (normality).

The first assumption is addressed by the principle of randomization used in the four experiments; all the measurements of one sample are not related to those of the other sample. The second and third assumptions are assessed by using the estimated residuals [2, 3]. To assess the homogeneity of variances, the Levene test for homogeneity of variances was applied [29]. The Levene test allowed to obtain a p-value of 0.7043, which suggests that variance in all treatment combinations (technique and program) are equal (null hypothesis of this test). Thus, the null hypotheses in favour of homogeneity were accepted. The third assumption (normality) was evaluated by applying the Kolmogorov-Smirnov test for normality [30,31]. After applying this test, a p-value of 0.2882 was obtained, which suggests that the residuals fit a normal distribution (null hypothesis of this test). Thus, the null hypothesis in favour of normality was accepted.

Once there is a valid statistical model, it is possible to draw reliable conclusions about the model components (technique, program and the interaction or relationship between technique and program). Table 12 shows the ANOVA results of the model stated in Equation (1).

If an α level of 0.05 is set, none of the components shows a significant difference with respect to efficiency. However, if the alpha level has the value of 0.1, which represents a confidence level of 90%, a significant difference is obtained with respect to efficiency in the program component. This suggests that at least one of the programs has a different level of efficiency than the others. To determine the significant difference the Tukey test for treatment comparisons was used [32]. Table 13 shows program comparisons with respect to efficiency.

As shown in Table 13, it can be observed that there is a significant difference with respect to efficiency between the banking and cmdline programs. This difference has an estimated value of 0.76 defects detected per hour, and suggests that the software program affects, to some degree, the efficiency of the three assessed software testing techniques, as shown in Figure 1.

Since the program component showed a significant difference with respect to efficiency (at $\alpha = 0.1$), it is important to estimate the extent by which efficiency is affected by a software

Table 12. Results of the analysis of variance (ANOVA)

Component	Df	Sum Sq	Mean Sq	F-value	p-value
Technique	2	0.418	0.2089	0.359	0.7003
Program	3	4.072	1.3574	2.335	0.0879
Technique: program	6	3.933	0.6555	1.128	0.3636
Residuals	41	23.835	0.5813		

Table 13. Pairwise comparisons with respect to the defect detection rates

Program comparisons	Difference	p-value
Banking – cmdline	0.7628	0.0621
Nametbl – cmdline	0.4558	0.4022
Ntree – cmdline	0.4581	0.3978
Nametbl – banking	−0.3069	0.7469
Ntree – banking	−0.3046	0.7512
Ntree – nametbl	0.0023	1.0000

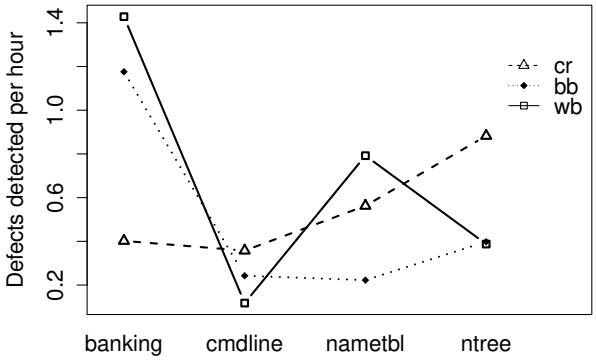

Figure 1. Interaction plot between the technique and the program

program type. Cohen's f was selected as the coefficient for assessing the average effect in the ANOVA program component across all its levels [33]. This coefficient can take values from zero to indefinitely large values. Cohen [33] suggests that values of 0.10, 0.25, and 0.40 represent small, medium, and large effect sizes, respectively. After estimating this coefficient, an effect size of $f = 0.41$ was obtained, which suggests a large effect size regarding the type of the used program.

With the effect size f estimated, it is possible to assess how sensitive (power test) any of the ANOVA components were in detecting an effect. We applied a post-hoc test to assess the degree of power achieved by the ANOVA program component. The power in a statistical test is equal to $1-\beta$, where β is the probability of making a Type II error. For program component we obtained a power of 0.79 (at α level = 0.1), which suggests that the acceptable level of power for the estimated effect size ($f = 0.41$) and the used sample size (53 collected measurements).

6.1. Defect analysis

In order to extend the previous efficiency analysis, the type of defects injected in the instrumented programswere scrutinized. The aim of this secondary analysis is to determine what classes of defects are detected by the studied testing techniques. As previously discussed in Section 5, defects were characterized by two classification schemes [5, 25]: Scheme 1 consisting of omission and commission defects, and scheme 2 consisting of cosmetic, initialization and control defects. Figure 2 shows the percentage of observed defects of the testing techniques (black-box [bb], white-box [wb] and code reading [cr]) split into the two defect classification schemes.

Figure 2 shows that participants using the code reading technique seem to observe more initialization defects than participants using the other techniques. Conversely, code reading and white-box seem to behave worse detecting cosmetic defects than black-box.

Similar to Figure 2, Figure 3 shows the percentage of observed defects by an instrumented program and by a defect type. As shown in Figure 3, all the techniques seem to produce worse results in the detection of cosmetic defectsthan black-box. Conversely, cosmetic defects injected in nametbl (na) and cmdline (cm) programs seem to negatively impact the percentage of observed defects.

Figures 2 and 3 give us an overview of the observed defects in these two schemes, however, an inferential analysis is needed to examine pos-

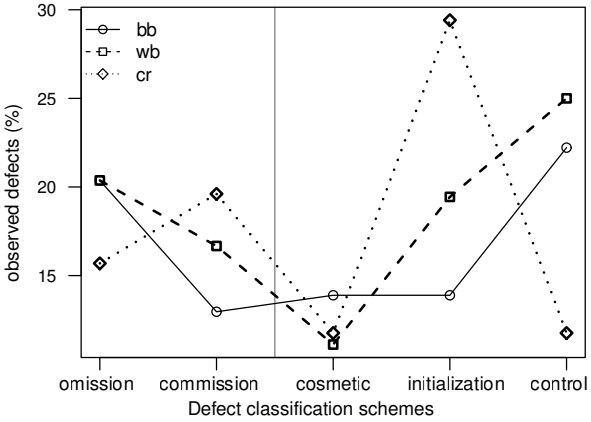

Figure 2. Types of defects observed by the testing technique

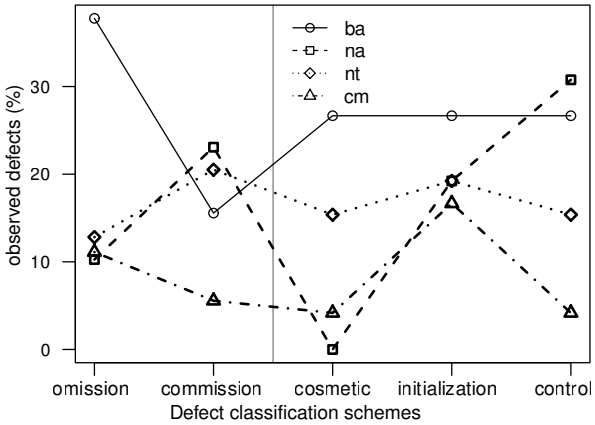

Figure 3. Types of defects observed by instrumented program

Table 14. Results of the analysis of variance (ANOVA) for defect classification scheme 1 (omission and commission defects)

Component	Df	Sum Sq	Mean Sq	F-value	p-value
Technique	2	62	30.9	0.052	0.9497
Program	3	4573	1524.2	2.548	0.0615
Scheme1	1	168	167.7	0.280	0.5979
Technique: program	6	3577	596.1	0.997	0.4334
Technique: scheme1	2	580	290.1	0.485	0.6175
Program: scheme1	3	5250	1749.8	2.925	0.0387
Tech.: prog.: scheme1	6	6086	1014.3	1.695	0.1325
Residuals	82	49056	598.2		

sible significant differences. Next the ANOVA results are presented according to the used defect classification schemes.

6.1.1. ANOVA results for defect classification scheme 1

The statistical model employed for this ANOVA is shown in Equation (2).

$$y_{ijkl} = \mu + \alpha_i + \beta_j + \gamma_k + (\alpha\beta)_{ij} + (\alpha\gamma)_{ik} + (\beta\gamma)_{jk} + (\alpha\beta\gamma)_{ijk} + \epsilon_{ijkl}\,. \quad (2)$$

In this equation μ is the grand mean; α_i represents the effect of software testing technique i, β_j represents the effect of program j, γ_k represents the effect of defect type k on the deffect classification scheme 1, $(\alpha\beta)_{ij}$ is the interaction effect between treatments i and j, $(\alpha\gamma)_{ik}$ is the interaction effect between treatments i and k, $(\beta\gamma)_{jk}$ is the interaction effect between treatments j and k, $(\alpha\beta\gamma)_{ijk}$ is the interaction effect between treatments i, j and k, l is the number of replications in each treatment combination, and ϵ is the random error which assumes $N(0, \sigma^2)$. The ANOVA results of this model are shown in Table 14.

As shown in Table 14 the program and the program:scheme1 components show a significant difference at alpha level of 0.1 and 0.05, respectively. To inspect the significant differences in these two components, the Tukey test for treatment comparisons was used [32]. Table 15 shows the pairwise comparisons of the program component and the interaction component (this between program and scheme 1).

A significant difference of 18% is observed between banking (ba) and cmdline (cm) programs. Participants applying the testing techniques observed more defects in the banking (ba) program. Another significant difference was observed between the omission defects of the banking program and the commission defects

Table 15. Significant pairwise comparisons for defect classification scheme 1

Pairwise comparisons	Difference (%)	p-value
Banking – cmdline	18.3333	0.0374
Banking: omission – cmdline: commission	32.3414	0.0213075
Nametbl: omission – banking: omission	−27.6415	0.0698521

of the cmdline program. Omission defects were the most commonly observed in these two programs. The third significant difference was observed between omission defects in the nametbl and banking programs, omission defects were the most commonly observed in the banking program.

Concerning model assumptions, the Levene test for homogeneity of variances [29] shows a non-significant p-value (0.9292), suggesting that variance in all treatment combinations (technique, program and defect classification scheme) are equal (the null hypothesis of this test). The assumption of normality was checked with the Kolmogorov-Smirnov test for normality [30, 31]. In this case the test showed a significant difference (p-value = 0.00012). Because measurements are represented as proportions (or percentages), these kinds of measurements can be prone to departures from normality, as shown in Figure 4.

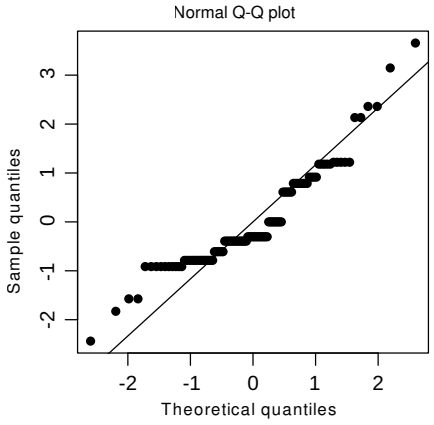

Figure 4. Normal Q-Q plot of standardized residuals given the model presented in Equation (2)

6.1.2. ANOVA results for defect classification scheme 2

Next the inferential analysis concerning the second defect classification scheme which is composed of cosmetic, initialization and control defect types is presented. Using the same statistical model as in Equation (2), but changing γ_k, representing now the effect of the defect type k of the defect classification scheme 2. Table 16 shows the results of the analysis of variance.

The results presented in Table 16 suggest a significant difference (at an alpha of 0.05) in the program component. The Tukey [32] was run to examine which of the program pairwise comparisons show significant differences. Table 17 shows the pairwise comparisons of the program component.

Table 17 suggests a significant difference of 18% between the banking (ba) and the cmdline (cm) program. Participants using the testing techniques observed a significantly larger number of defects in the banking program than in the cmdline program.

In relation to the model assumptions, the Levene test for homogeneity of variances [29] shows a non-significant p-value of 0.7501 in favor of the equality variances among treatments, however, in the same way as in the previous analysis, some departures from normality were observed with the Kolmogorov-Smirnov test [30, 31], a significant p-value of 0.00012 was observed.

7. Network meta-analysis

The results of the replication reported here can be incorporated in the existing evidence of related experiments. With the quantitative information available in similar experiments [5–8, 11] it is possible to carry out a network meta-analysis in order to offer better informed decisions on the efficiency of the testing techniques reported in previous experiments along with the one discussed in this work.

The network meta-analysis approach (NMA) [34, 35], also known as multiple treatment comparison or mixed treatment comparison, has

Table 16. Results of the analysis of variance (ANOVA) for scheme 2 (cosmetic, initialization and control defects)

Component	Df	Sum Sq	Mean Sq	F-value	p-value
Technique	2	93	46.3	0.055	0.946
Program	3	6859	2286.3	2.727	0.047
Scheme2	2	2296	1147.8	1.369	0.258
Technique: program	6	5365	894.2	1.067	0.386
Technique: scheme2	4	3826	956.6	1.141	0.340
Program: scheme2	6	5224	870.7	1.039	0.404
Tech.: prog.: scheme2	12	3904	325.3	0.388	0.966
Residuals	123	103125	838.4		

Table 17. Significant pairwise comparisons for defect classification scheme 2

Pairwise comparisons	Difference (%)	p-value
Banking – cmdline	18.3333	0.0274

been increasingly widespread in recent years in the health care arena [36–38].

The network meta-analysis approach can integrate direct and indirect evidence in a collection of studies (or experiments). This approach provides information on the relative effects of three or more treatments for the same outcome [39]. Conversely to classical meta-analysis, NMA simultaneously compares the effects of three or more treatments.

Given the evidence of the present replication (pooling together the four experiments as one experiment replication) along with the evidence of seven related experiments [5–8,11], NMA with the 'netmeta' R package [40] was performed to assess the available evidence of the efficiency of the testing techniques: black-box (bb), white-box (wb) and code reading (cr). Table 18 shows the sample sizes (n), average defect detection rates (mean) and the standard deviations (sd) of the three testing techniques examined in the aforementioned experiments.

With the information available in Table 18 it is possible to carry out NMA. Asa result of conducting all these experiments to examine the same three testing techniques it can be concluded that they conform to a single design providing only direct evidence. NMA can also be applied to estimate indirect evidence, however, not in this case. For example, suppose there are experiments examining treatments A and B and experiments examining treatments A and C (for the same outcome), these experiments can be pooled together in NMA to obtain an indirect estimate for indirect comparison between treatments B and C by means of a common comparator, that is treatment A.

Table 19 shows the resulting NMA obtained on the basis of the information of Table 18. The results are presented in the matrix of estimated overall effect sizes (with lower and upper confidence limits) belonging to all the pairwise treatment comparisons. Effect sizes were computed using the standardized mean difference (Hedges' g) [41]. The guidelines proposed by Cohen [33] suggest that effect sizes of 0.2, 0.5 and 0.8 represent small, medium and large effect sizes, respectively. Due to possible context differences in these experiments, the effect sizes shown in Table 19 were estimated according to a random effects model, assuming that the underlying effects in the experiments of the same treatment comparison come from a common normal distribution, i.e. an account for unexplained heterogeneity was assumed.

To obtain valid conclusions from NMA, the resulting network of treatments should be assessed against the transitivity and consistency assumptions [42–44]. In the case of the transitivity assumption, the network is assumed to maintain transitivity whenever pairwise treatment

Table 18. Sample sizes, average defect detection rates and standard deviations of the testing techniques examined in eight experiments

Experiment	n_{bb}	$mean_{bb}$	sd_{bb}	n_{wb}	$mean_{wb}$	sd_{wb}	n_{cr}	$mean_{cr}$	sd_{cr}
umd82 [5]	29	1.58	0.90	29	1.40	0.87	29	1.90	1.83
umd83 [5]	13	1.22	0.91	13	1.18	0.84	13	0.56	0.46
umd84 [5]	32	1.84	1.06	32	1.82	1.24	32	3.33	3.42
ukl94 [6, 7]	27	4.67	2.27	27	2.92	1.59	27	2.11	1.12
ukl95 [6, 7]	21	3.08	1.28	18	2.00	1.59	17	1.74	0.67
uos97 [8]	47	2.47	1.10	47	2.20	0.94	47	1.06	0.75
uok11 [11]	18	2.46	0.58	18	2.50	0.83	18	2.16	0.55
epch14	18	0.54	0.60	18	0.73	1.06	17	0.54	0.64

Table 19. Pairwise treatment overall effect size estimates, lower and upper 95% confidence limits under a random effects model

Technique	Black-box (bb)	Code reading (cr)	White-box (wb)
Black-box (bb)	–	0.576 (0.110, 1.042)	0.239 (−0.224, 0.701)
Code reading(cr)	−0.576 (−1.042, −0.110)	–	−0.337 (−0.802, 0.127)
White-box (wb)	−0.239 (−0.701, 0.224)	0.337 (−0.127, 0.802)	–

effects are similarly distributed across the studies (experiments). For example, suppose some studies assessing treatments A, B, C for the same outcome, if treatment A performs better than B, and treatment B performs better than C, then treatment A has to perform better than C (transitivity is met). Departures from transitivity arise when significant heterogeneity is present across one or more pairwise treatment comparisons in the network. On the other hand, the consistency assumption states that both direct and indirect evidence in a given pairwise treatment comparison (network edge) should be similar. This assumption only applies to situations where there is both direct and indirect evidence in one or more edges of the network [42].

In a similar way when the Q statistic is used in pairwise meta-analysis, a generalization of such index is used in NMA. In NMA, the Q statistic measures the deviation from heterogeneity/inconsistency. Index Q can be separated into parts for each pairwise treatment comparison and a part for the remaining inconsistency between all the treatment pairwise comparisons [43].

Given the resulting NMA in Table 19, the statistical test for assessing the heterogeneity/inconsistency of the network is run. In the same manner as in pairwise meta-analysis, in NMA the used Q statistic follows a Chi-Squared distribution. The test showed a Q-value of 74.12, corresponding with a significant p-value smaller than 0.0001, thus suggesting a significant degree of heterogeneity in the network. The I^2 index that represents the percentage of heterogeneity also showed a high value of 81.1%.

The heterogeneity found in the network suggests that at least one pairwise treatment comparison contains contradictory effect size estimates, yielding a significant heterogeneity in the network edge. Because of this situation, it was decided to assess the heterogeneity (under classical meta-analysis also using Hedges' g [41]) in each network edge, i.e. with the following pairwise comparisons: black-box (bb) vs. code reading (cr), black-box (bb) vs. white-box (wb) and code reading (cr) vs. white-box (wb). Table 20 shows the Q and I^2 coefficients in each network edge.

Table 20. Assessment of heterogeneity in each network edge

Edge	Q	p-value	I^2
bb, cr	60.81	<0.0001	88.5%
bb, wb	11.40	0.1221	38.6%
cr, wb	42.40	<0.0001	83.5%

According to Table 20 the pairwise comparison between black-box and white-box yields con-

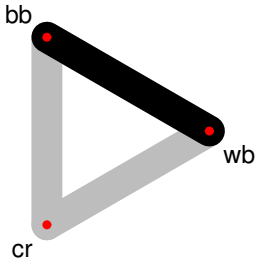

Figure 5. Network graph with a consistent edge highlighted

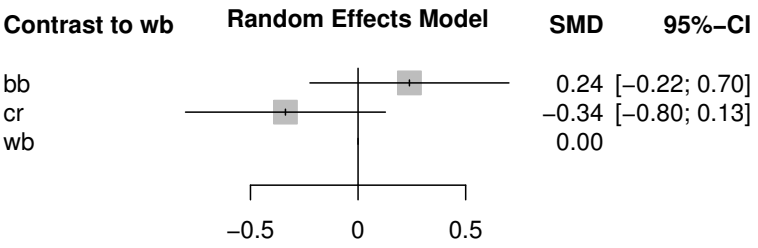

Figure 6. Forest plot with the overall effect size estimates and confidence limits of the testing techniques

sistent results (p-value is non-significant) suggesting a degree of homogeneity among effect size estimates of the eight experiments. The resulting I^2 coefficient indicates that experiments in this edge present a low level of heterogeneity which is non-significant. As observed in Table 20, the rest of pairwise comparisons show a significant difference. Figure 5 shows the resulting network graph with the pairwise treatments. The network is laid out in a plane where nodes correspond to the treatments (bb, wb and cr) whereas edges represent the pairwise treatment comparisons; the observed consistent edge is highlighted in black. The thickness of the lines represents the number of experiments available for each treatment, in this case, eight experiments.

Figure 6 displays a forest plot of pairwise overall effect size estimates using the white-box technique as the reference treatment. As it was discussed ealier, only the pairwise comparison between black-box and white-box shows homogeneity in its effect size estimates. It is visible that a small effect size of 0.24 is observed in favour of the black-box technique, however, the estimated confidence limits indicate that the overall effect size could be zero, thus suggesting that both black-box and white-box yield similar defect detection rates.

8. Discussion

Having presented the analysis, in this section the findings in reference to the research questions stated and previous work are discussed.

According to the evidence collected in the four experiments, all the testing techniques showed similar levels of efficiency (no significant difference was observed); this suggests that efficiency is not affected by the testing techniques **(RQ1)**. Evidence suggesting that the type of used software program affects the efficiency of the studied testing techniques has been found **(RQ2)**. In addition, the current evidence suggests that efficiency is not affected by the relationship between techniques and software programs, i.e. both factors are independent **(RQ3)**.

In relation to the baseline experiment, the results support the results presented in experiments umd82 and umd83 [5], the three testing techniques behave in a similar way. With regard to the efficiency of testing techniques, Table 21 shows a comparison between the reported results in [5] and the pooled results. These results indicate the average number of defects detected per hour.

As shown in Table 21, this replication supports the results of the umd82 and umd83 experiments [5]. In these experiments, the null hypothesis is accepted because the three testing techniques do not show significant differences in the defect detection rates. However, this difference is significant in umd84 [5]. One possible reason for this significant difference could be the participants' expertise. In the third experiment reported in [5], participants were programming professionals with high levels of technical skill.

As noted in Table 21, the obtained rates of efficiency were lower than those reported in [5], perhaps the number of defects injected in the instrumented programs (six per program) or the participants' expertise yielded a lower efficiency rate. One point worth noting about the umd82 and umd83 experiments in [5] is that although similar kinds of participants were used, the efficiency measurements in the first experiment are

Table 21. Average defect detection rates per a testing technique reported in [5] and in replication

Technique	umd82 [5]	umd83 [5]	umd84 [5]	epch14
Code reading	1.90	0.56	3.33	0.54
Black-box	1.58	1.22	1.84	0.54
White-box	1.40	1.18	1.82	0.73

slightly higher than those in the second experiment. One possible reason for this is that both the software type and the injected defects affected the efficiency. In the three experiments reported in [5], the authors did not use the same instrumented programs in all experiments. It is also important to note that in [5], each program had a different number of defects with respect to both defect classification schemes; i.e. the used programs were not equally balanced regarding the number of defects and defect types. It is highly probable that differences in the program type and the used defects produced an interaction effect with the testing techniques, as mentioned in [5]. With the aim of avoiding this effect interaction, the same number and the same type of defects were employed in the instrumented programs, and only the type od software program varied.

With respect to the average defect detection rate per program, the obtained results support those in umd82 and umd84 [5], the type of software impacts on the efficiency of the testing techniques. Table 22 shows the average defect detection rates of umd82, umd83, umd84 and the results obtained here, the efficiency seems to vary depending on the software type.

It is possible that the low rates of efficiency observed in the replication are due to the expertise level of the participants. According to our evidence, it seems that participants had a better understanding of the domain related to the banking instrumented program than the domains related to the other programs. Cyclomatic complexity is discarded as a possible factor that could affect efficiency. Programs nametbl, ntree and cmdline have similar levels of VG (43, 46 and 45, respectively), however, cmdline still showed the worst efficiency rate. This evidence seems to reinforce the idea that the knowledge of the program domain could affect the efficiency of software testing techniques.

Concerning the defect analysis (secondary research question, **SRQ1**), no significant differences in the two defect classification schemes were observed in the case of the testing techniques. The results in the baseline experiment [5] suggest that participants applying code reading and those applying black-box observed significantly more omission defects than those applying white-box. In the case of the conducted experiments no significant difference between omission and commission defects was observed.

With regard to the second defect classification scheme the authors in [5] observed that participants using code reading and those using black-box observed significantly more initialization defects than those using white-box. Participants using code reading observed significantly more interface defects than those using either black-box or white-box. Participants using black-box observed significantly more control defects than those using the other two techniques. Participants using code reading observed significantly more computation defects than those using white-box. With regard to data and cosmetic defects the authors in [5] did not observe significant differences. In the case of the experiments described here, it was observed that code reading seemed to detect more initialization defects than white-box and black-box but the difference was not significant. It was also found out that white-box and black-box seemed to detect more control defects than code reading, but again, this difference was not significant.

However, with respect to the instrumented programs, significant differences were observed, more omission defects were detected in the banking program compared to cmdline program (defect classification scheme 1). Similarly there were more cosmetic, initialization and control defects (defect scheme 2) observed in the banking program than in the cmdline program. Although

Table 22. Average defect detection rates per program reported in [5] and in replication

Program	umd82 [5]	umd83 [5]	umd84 [5]	epch14
Formatter	1.60	0.98	2.15	–
Plotter	1.19	0.92	–	–
Data type	2.09	–	3.70	–
Database	–	1.05	1.14	–
Banking	–	–	–	1
Nametbl	–	–	–	0.55
Ntree	–	–	–	0.54
Cmdline	–	–	–	0.24

same defect types (for both schemes) were equally injected in the four instrumented programs, they were observed in a similar way, perhaps these findings suggest that the knowledge of the domain of the program to be tested may impact the efficiency of the testing techniques.

The replication results along with the related existing ones allowed to carry out a quantitative synthesis using the network meta-analysis (NMA) approach. Taking into account the quantitative information of eight experiments run in five countries (USA, Germany, UK, India and Ecuador) only consistent results were observed among black-box and white-box techniques, both techniques yielded similar efficiency rates. The code reading technique compared with either black-box or white-box techniques showed inconsistent results (presence of heterogeneity). These results suggest that the code reading technique shows a greater level of sensitivity. In umd84 [5], code reading was significantly more efficient than black-box and white-box, probably in this experiment the expertise had an impact on this technique. In the umd84 experiment, participants were programming professionals with an overall average of ten years of professional experience. However, the expertise factor does not seem to significantly affect the efficiency of the black-box and white-box, in this treatment comparison (bb vs. wb), participants used in the pooled experiments spanned different expertise levels, such as undergraduate, graduate and professionals. In the case of the other treatment comparisons (cr vs. bb, and cr vs. wb), further subgroup analyses [45, 46] can be performed to identify moderators affecting the outcomes.

For the purpose reporting this experiment replication, the guidelines of [19] were taken as reference, however, it is not so clear how to proceed when reporting a replication in the context of a family of related experiments. In this sense, the authors propose that the family of experiments be explained in terms of the main treatments studied along with the contextual information. It is also proposed that the findings of the family be organized, first, according to the cause and effect constructs and, second, according to the cause and effect operationalizations that the related experiments address. The findings of the family can be presented as a narrative or quantitative synthesis (or both of them).

The authors also propose to describe the baseline experiment, this activity underlies the essence of the replication, which refers to the repetition of a previously run experiment [14], it is recomended to describe the main findings and contextual information of the baseline experiment. Once the family and the baseline have been described, the replication and the analysis of the results should be described. Next, the replication findings should be analyzed in relation to the baseline experiment along with the related experiments of the family. Depending on the type of analysis done, this activity can be performed as a qualitative or quantitative synthesis (or both of them). Finally, a discussion of the consistency or inconsistency of the findings should be addressed.

Regarding the limitations of the experiment replication reported here, below the strategies used to minimize the threats to validity are described [47]. With respect to conclusion validity, the measurements collected in these experiment sessions satisfy the principles of independence, homogeneity and normality. With respect to internal validity, participants were randomly assigned to treatments, which reduced learning effects. Boredom or fatigue was reduced by using alternate sessions. Participants were in the same classroom, working under the same conditions, and sitting apart with no interaction. With respect to construct validity, cause and effect constructs were operationalized in the same way as is reported in [5] and in [9]. With respect to external validity, the use of students instead of practitioners might have compromised this type of validity. However, there exists some evidence suggesting that in some contexts, the results of empirical studies that employ students with enough technical skills are equivalent to the results of empirical studies that use practitioners [48]. For example, using students in their last academic year of an undergraduate program as experiment participants may be comparable to using junior practitioners as participants. In fact, it is common for students in their last academic year to work part-time in IT-related companies. In this sense, the results presented here may be generalizable to junior practitioners.

9. Conclusions

In this work the efficiency of three software testing techniques has been assessed. The replication was composed of four experiments where several instrumented software programs were used. The obtained results suggest that software testing techniques perform in a similar way, but the domain related to the software to be tested might have an effect on the defect detection rates of the testing techniques. We suggest that software verification activities such as software testing be performed only after software engineers have a clear understanding of the software product domain.

The main contributions of this work are the following: 1) the execution of a controlled experiment replication in order to verify previous findings, and 2) the realization of a quantitative synthesis with the aim of consolidating the findings belonging to a family of related experiments.

Acknowledgements

This research study received support from the Prometeo project 20140697BP funded by the Government of the Republic of Ecuador's Department of Higher Education, Science, Technology and Innovation (Senescyt). Special thanks to Jahzeel J. Coss who developed and instrumented the programs used in the experiment replication reported here. César Pardo acknowledges the contribution of the University of Cauca, where he works as an assistant professor.

References

[1] S. McConnell, *Code Complete*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2004.

[2] G.E.P. Box, W.G. Hunter, J.S. Hunter, and W.G. Hunter, *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. John Wiley & Sons, Jun. 1978.

[3] R. Kuehl, *Design of Experiments: Statistical Principles of Research Design and Analysis*, 2nd ed. California, USA: Duxbury Thomson Learning, 2000.

[4] N. Juristo and A.M. Moreno, *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.

[5] V. Basili and R. Selby, "Comparing the effectiveness of software testing strategies," *IEEE Trans. Softw. Eng.*, Vol. 13, No. 12, 1987, pp. 1278–1296.

[6] E. Kamsties and C.M. Lott, *An Empirical Evaluation of Three Defect-Detection Techniques*. Berlin, Heidelberg: Springer, 1995, pp. 362–383.

[7] E. Kamsties and C. Lott, "An empirical evaluation of three defect detection techniques," Dept. Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, Tech. Rep. ISERN 95-02, 1995.

[8] M. Roper, M. Wood, and J. Miller, "An empirical evaluation of defect detection techniques," *Information and Software Technology*, Vol. 39, No. 11, 1997, pp. 763–775.

[9] N. Juristo and S. Vegas, "Functional testing, structural testing and code reading: What fault type do they each detect?" in *Empirical Methods and Studies in Software Engineering*, ser. Lecture Notes in Computer Science, R. Conradi and A. Wang, Eds. Berlin, Heidelberg: Springer, 2003, Vol. 2765, pp. 208–232.

[10] N. Juristo, S. Vegas, M. Solari, S. Abrahao, and I. Ramos, "Comparing the effectiveness of equivalence partitioning, branch testing and code reading by stepwise abstraction applied by subjects," in *IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, Apr. 2012, pp. 330–339.

[11] S.U. Farooq and S. Quadri, "An externally replicated experiment to evaluate software testing methods," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '13. New York, NY, USA: ACM, 2013, pp. 72–77.

[12] O.S. Gómez, R.A. Aguilar, and J.P. Ucán, "Efectividad de técnicas de prueba de software aplicadas por sujetos novicios de pregado," in *Encuentro Nacional de Ciencias de la Computación, (ENC)*, M.D. Rodríguez, A.I. Martínez, and J.P. García, Eds., Ocotlán de Morelos, Oaxaca, México, Nov. 2014.

[13] O.S. Gómez, N. Juristo, and S. Vegas, "Understanding replication of experiments in software engineering: A classification," *Information and Software Technology*, Vol. 56, No. 8, 2014, pp. 1033–1048.

[14] N. Juristo and O.S. Gómez, "Replication of software engineering experiments," in *Empirical Software Engineering and Verification: LASER Summer School 2008–2010*, ser. Lecture Notes in Computer Science, B. Meyer and M. Nordio, Eds. Berlin, Heidelberg: Springer, Nov. 2011, Vol. 7007, pp. 60–88.

[15] O.S. Gómez, "Tipología de replicaciones para la síntesis de experimentos en ingeniería del software," Ph.D. dissertation, Facultad de Informática de la Universidad Politécnica de Madrid, Campus de Montegancedo, 28660, Boadilla del Monte, Madrid, España, May 2012.

[16] D. Sjøberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N.K. Liborg, and A. Rekdal, "A survey of controlled experiments in software engineering," *Software Engineering, IEEE Transactions on*, Vol. 31, No. 9, Sep. 2005, pp. 733–753.

[17] F. da Silva, M. Suassuna, A. França, A. Grubb, T. Gouveia, C. Monteiro, and I. dos Santos, "Replication of empirical studies in software engineering research: A systematic mapping study," *Empirical Software Engineering*, Vol. 19, No. 3, 2014, pp. 501–557.

[18] R.C. Linger, B.I. Witt, and H.D. Mills, *Structured Programming; Theory and Practice the Systems Programming Series*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1979.

[19] J. Carver, "Towards reporting guidelines for experimental replications: A proposal," in *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER)*, Cape Town, South Africa, May 2010.

[20] W. Howden, "Functional program testing," *IEEE Transactions on Software Engineering*, Vol. 6, 1980, pp. 162–169.

[21] G.J. Myers, *The Art of Software Testing*. New York: John Wiley & Sons, 1979.

[22] B. Marick, *The craft of software testing: subsystem testing including object-based and object-oriented testing*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[23] B. Beizer, *Software testing techniques*, 2nd ed. New York, NY, USA: Van Nostrand Reinhold Co., 1990.

[24] H.L. Dreyfus and S. Dreyfus, *Mind over Machine. The Power of Human Intuition and Expertise in the Era of the Computer*. New York: Basil Blackwell, 1986.

[25] V. Basili and B. Perricone, "Software errors and complexity: An empirical investigation," *Commun. ACM*, Vol. 27, No. 1, 1984, pp. 42–52.

[26] V. Basili, G. Caldiera, and H. Rombach, "Goal question metric paradigm," in *Encyclopedia of Software Engineering*, J.J. Marciniak, Ed. Wiley-Interscience, 1994, pp. 528–532.

[27] P. Louridas and G. Gousios, "A note on rigour and replicability," *SIGSOFT Softw. Eng. Notes*, Vol. 37, No. 5, Sep. 2012, pp. 1–4.

[28] O.S. Gómez, N. Juristo, and S. Vegas, "Replication, reproduction and re-analysis: Three ways for verifying experimental findings," in *International Workshop on Replication in Empirical Software Engineering Research (RESER)*, Cape Town, South Africa, May 2010.

[29] H. Levene, "Robust tests for equality of variances," in *Contributions to probability and statistics*, I. Olkin, Ed. Palo Alto, CA: Stanford Univ. Press, 1960, pp. 278–292.

[30] A.N. Kolmogorov, "Sulla determinazione empirica di una legge di distribuzione," *Giornale dell'Istituto Italiano degli Attuari*, Vol. 4, 1933, pp. 83–91.

[31] N.V. Smirnov, "Table for estimating the goodness of fit of empirical distributions," *Ann. Math. Stat.*, Vol. 19, 1948, pp. 279–281.

[32] J. Tukey, "Comparing individual means in the analysis of variance," *Biometrics*, Vol. 5, No. 2, 1949, pp. 99–114.

[33] J. Cohen, *Statistical power analysis for the behavioral sciences*. Hillsdale, NJ: L. Erlbaum Associates, 1988.

[34] T. Lumley, "Network meta-analysis for indirect treatment comparisons," *Statistics in Medicine*, Vol. 21, No. 16, 2002, pp. 2313–2324.

[35] G. Lu and A.E. Ades, "Combination of direct and indirect evidence in mixed treatment comparisons," *Statistics in Medicine*, Vol. 23, No. 20, 2004, pp. 3105–3124.

[36] T. Greco, G. Biondi-Zoccai, O. Saleh, L. Pasin, L. Cabrini, A. Zangrillo, and G. Landoni, "The attractiveness of network meta-analysis: A comprehensive systematic and narrative review," *Heart, Lung and Vessels*, Vol. 7, No. 2, 2015, pp. 133–142.

[37] A. Bafeta, L. Trinquart, R. Seror, and P. Ravaud, "Reporting of results from network meta-analyses: Methodological systematic review," *BMJ*, Vol. 348, 2014. [Online]. http://www.bmj.com/content/348/bmj.g1741

[38] A. Nikolakopoulou, A. Chaimani, A.A. Veroniki, H.S. Vasiliadis, C.H. Schmid, and G. Salanti, "Characteristics of networks of interventions: A description of a database of 186 published networks," *PLoS ONE*, Vol. 9, No. 1, Dec. 2014, pp. 1–10.

[39] A. Chaimani and G. Salanti, "Visualizing assumptions and results in network meta-analysis: The network graphs package," *Stata Journal*, Vol. 15, No. 4, 2015, pp. 905–950.

[40] G. Rücker, G. Schwarzer, U. Krahn, and J. König, *netmeta: network Meta-Analysis using Frequentist Methods*, 2016, R package version 0.9-0. [Online]. https://CRAN.R-project.org/package=netmeta

[41] L.V. Hedges and I. Olkin, *Statistical methods for meta-analysis*. Orlando: Academic Press, 1985.

[42] F. Song, Y.K. Loke, T. Walsh, A.M. Glenny, A.J. Eastwood, and D.G. Altman, "Methodological problems in the use of indirect comparisons for evaluating healthcare interventions: Survey of published systematic reviews," *BMJ*, Vol. 338, 2009.

[43] J.P.T. Higgins, D. Jackson, J.K. Barrett, G. Lu, A.E. Ades, and I.R. White, "Consistency and inconsistency in network meta-analysis: Concepts and models for multi-arm studies," *Research Synthesis Methods*, Vol. 3, No. 2, 2012, pp. 98–110.

[44] J.P. Jansen and H. Naci, "Is network meta-analysis as valid as standard pairwise meta-analysis? it all depends on the distribution of effect modifiers," *BMC Medicine*, Vol. 11, May 2013, pp. 159–159.

[45] M. Borenstein, L.V. Hedges, J.P. Higgins, and H.R. Rothstein, *Introduction to Meta-Analysis*. United Kingdom: John Wiley & Sons, Ltd, 2009.

[46] M. Ciolkowski, "What do we know about perspective-based reading? An approach for quantitative aggregation in software engineering," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 133–144.

[47] T. Cook and D. Campbell, *The design and conduct of quasi-experiments and true experiments in field settings*. Chicago: Rand McNally, 1976.

[48] P. Runeson, "Using students as experiment subjects – an analysis on graduate and freshmen student data," in *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering*, Keele University, UK, 2003, pp. 95–102.

Appendix: Experiment replication raw data

In this appendix we provide the measurements collected in our experiment replication. Table A shows the defects observed and the time that participants spent applying the testing techniques.

Table A. Observed defects and time spent applying the testing techniques

Case	Technique	Program	F1	F2	F3	F4	F5	F6	Minutes
1348	white-box	cmdline					•		127
1349	white-box	cmdline							94
1350	white-box	cmdline							116
1351	white-box	cmdline							280
1343	black-box	cmdline							106
1342	black-box	cmdline						•	120
1344	black-box	cmdline							105
1345	black-box	cmdline						•	127
1355	code reading	cmdline							79
1356	code reading	cmdline						•	84
1358	code reading	cmdline							163
1357	code reading	cmdline				•		•	166
1291	white-box	banking	•					•	109
1292	white-box	banking			•				120
1293	white-box	banking							219
1294	white-box	banking			•			•	109
1295	white-box	banking	•		•		•	•	54
1285	black-box	banking			•				110
1286	black-box	banking	•	•			•	•	106
1287	black-box	banking	•		•				132
1288	black-box	banking			•		•		110
1289	black-box	banking			•				56
1297	code reading	banking							138
1298	code reading	banking		•		•	•	•	154
1299	code reading	banking						•	134
1300	code reading	banking							137
1301	code reading	banking							87
1312	white-box	nametbl	•			•			94
1310	white-box	nametbl	•			•			87
1311	white-box	nametbl							142
1314	white-box	nametbl	•		•				115
1313	white-box	nametbl				•			230
1306	black-box	nametbl	•		•				135
1304	black-box	nametbl							146
1305	black-box	nametbl							134
1307	black-box	nametbl							134
1316	code reading	nametbl			•	•		•	114
1319	code reading	nametbl	•						89
1317	code reading	nametbl							72
1318	code reading	nametbl							250
1329	white-box	ntree	•	•					116
1331	white-box	ntree							141
1330	white-box	ntree				•			115
1332	white-box	ntree							285
1323	black-box	ntree						•	141
1324	black-box	ntree		•				•	107
1327	black-box	ntree							113
1325	black-box	ntree				•			137
1326	black-box	ntree							108
1336	code reading	ntree							85
1335	code reading	ntree			•	•			112
1338	code reading	ntree				•	•	•	97
1337	code reading	ntree		•					100