Variation Factors in the Design and Analysis of Replicated Controlled Experiments - Three (Dis)similar Studies on Inspections versus Unit Testing

Runeson, Per; Stefik, Andreas; Andrews, Anneliese

Published in:
Empirical Software Engineering

DOI:
10.1007/s10664-013-9262-z

2014

Variation Factors in the Design and Analysis of Replicated Controlled Experiments

– Three (Dis)similar Studies on Inspections versus Unit Testing

Per Runeson, Andreas Stefik and Anneliese Andrews
Lund University, Sweden, per.runeson@cs.lth.se
University of Nevada, NV, USA, stefika@gmail.com
University of Denver, CO, USA, andrews@cs.du.edu

Empirical Software Engineering, DOI: 10.1007/s10664-013-9262-z
Self archiving version. The final publication is available at link.springer.com

August 13, 2013

Abstract

Background. In formal experiments on software engineering, the number of factors that may impact an outcome is very high. Some factors are controlled and change by design, while others are are either unforeseen or due to chance. **Aims.** This paper aims to explore how context factors change in a series of formal experiments and to identify implications for experimentation and replication practices to enable learning from experimentation. **Method.** We analyze three experiments on code inspections and structural unit testing. The first two experiments use the same experimental design and instrumentation (replication), while the third, conducted by different researchers, replaces the programs and adapts defect detection methods accordingly (reproduction). Experimental procedures and location also differ between the experiments. **Results.** Contrary to expectations, there are significant differences between the original experiment and the replication, as well as compared to the reproduction. Some of the differences are due to factors other than the ones designed to vary between experiments, indicating the sensitivity to context factors in software engineering experimentation. **Conclusions.** In aggregate, the analysis indicates that reducing the complexity of software engineering experiments should be considered by researchers who want to obtain reliable and repeatable empirical measures.

1 Introduction

Replications are a cornerstone of the empirical sciences, whether referring to the general definition of testing the same hypothesis in different studies or the narrow meaning of repeating the same experimental procedures [44]. Cartwright refers to *replications* as repeating an experiment, closely following the experimental procedures of the original ones, and refers to *reproductions* when reexamining the results from a previous experiment, using a different experimental protocol [5, 26]. Possible benefits of replication include an assessment of the confidence level for the results of the original experiment, to improve the internal validity and reliability of the conclusions, while their generalizability are studied by reproductions, improving the external validity [37]. Replications conducted by different researchers can also be used to study the effect of researcher bias or other potentially confounding factors. While the need for and value of replications is increasingly accepted in the empirical software engineering community, evidence on how similar or dissimilar experiments ideally and practically should be is insufficiently covered in the literature, i.e. the borderlines between *replications* and *reproductions* and the expected contribution from the two types of repeated experiments is unclear.

Consider that as it currently stands in the literature, an established and empirically grounded taxonomy of types of replications does not exist, neither in software engineering [12], nor in other fields of science [44]. Thus, several terms are used to qualify replication types, like *direct, exact, conceptual, independent* etc., or *internal* versus *external*, depending on whether the same or a different group of researchers performed a replication. The terms *replication* or *reproduction*, as we use them in this paper, also exist, although the definition of each term is not universally agreed upon. Second, there are diverging positions on whether replications should be as exact as possible or vary factors, like experimenters, instrumentations, location, instantiation and parametrization of techniques etc. [46, 25]. Juristo and Vegas concluded that"After numerous attempts over the years, apart from experiments replicated by the same researchers at the same site, no exact replications have yet been achieved." [25]; a position also supported by statisticians [33]. Third, in an attempt to control the variation factors, Clarke and O'Connor developed a comprehensive framework for context factors, comprising 44 factors and 170 sub-factors [6]. However, as insightfully noted by Dybå *et al.* [10], even if each factor had only two factor levels, that generates 2^{170} combinations, which is more than the number of atoms in the world! Apparently, we have to proceed along other routes than exhaustive exploration of factorial combinations of context factors in replications.

In order to explore these issues on similarity between replications and factors varying between them on purpose, or by chance, this paper analyzes three experiments comparing structural unit testing to code inspections. A large body of experiments with various forms of code inspection and unit testing methods exists, beginning with Hetzel's work in the early 1970's [14], continuing with Basili *et al.*'s well cited "family of experiments" [3]. Despite attempts to synthesize the findings from these studies [22, 23, 40], and to conduct meta-analyses

[35], there is still no unified understanding of the results. This may be due to a number of *potential* factors, including at least: 1) variations between replications that confuse the synthesized results, 2) insufficient power of the individual studies, and 3) inherent weaknesses of the experimental designs.

We analyze three experiments to explore these issues, two of which were designed to be as similar as possible (replication), with one designed to be different (reproduction). The first experiment [41] compared usage-based inspection and branch testing. This experiment could not reject the null hypothesis related to defect detection rate, but found a statistically significant difference between techniques related to the number of defects detected. The second experiment, run by the same researchers, in the same context, with the same instruments (replication), investigated this issue further. However, this subject group was smaller and the time spent by one of the groups was significantly higher; giving rise to a new factor. In this case the defect detection rate was significantly higher for the technique which spent the least time, while the total number of defects found were not different between the techniques. The third experiment, conducted by a different group of researchers, expands the investigation by using software for embedded systems as the objects and consequently different instantiations of the defect detection techniques (reproduction). The third experiment provided significantly different results for the two techniques, reversing which technique was best, compared to the original and replication experiments.

Neither of the follow-up experiments were published separately; instead, we presented them in conjunction with a preliminary meta-analysis [43], which is further extended in this paper. We use these experiments to analyze the impact of factors on replications and reproductions and extend previous work to identify the implications for experiment planning and reporting practices for replications in general.

The paper starts with some background, relevant definitions and related work in Section 2. Section 3 describes experiment planning and operations. Section 4 analyzes and compares the results of the experiments in isolation, and Section 5 analyzes across the experiments. Section 6 discusses implications for the research community and Section 7 concludes the paper.

2 Background and Related Work

2.1 Replication types

Replications and analyses across several studies has been discussed for long, for example, by Pickard *et al.* [38], Miller [35], Kitchenham [30], Shull *et al.* [45, 46]. Recently, da Silva *et al.* [8] systematically compiled replications in software engineering. They found 96 papers, reporting 133 replications of 72 original studies, conducted between 1994 and 2010. Interestingly, they found that *internal* replications tend to confirm results while *external* replications largely do not. Internal replications confirmed the results from the original experiment in 82% of the cases and confirmed them partly in 9%, while external replications only

confirmed the original results in 26% of the cases, and confirmed them partly in 28% of the cases. da Silva *et al.* hypothesize the differences: 1) intentional and unintentional variations are more likely to appear in external replications, 2) differences in experimental context between the original and the replication site may have an impact, and 3) external researchers may lack tacit knowledge about the original study design or analysis.

In regards to the reuse of experimental material, Miller [35] states, "although from a simple replication point of view, this seems attractive, from a meta-analysis point of view this is undesirable, as it creates strong correlations between the two studies". Pickard *et al.* comments on the other hand on the outcome of the primary studies [38] "the greater the degree of similarity between the studies the more confidence you can have in the results of a meta-analysis". Hence there is a conflict between what is desirable from a statistical point of view and from a learning point of view.

Gomez *et al.* [12] surveyed several fields of research, including social science, business and philosophy topics, and identified three major groups of replications. For each of the groups, they collected 10–20 variations of terms for the type of replication. They identified three major groups of replications:

1. Replications that vary little or not at all with respect to the reference experiment.

2. Replications that do vary but still follow the same method as the reference experiment.

3. Replications that use different methods to verify the reference experiment results.

Shull *et al.* [46] distinguish between "*exact* replications, in which the procedures of an experiment are followed as closely as possible; and *conceptual* replications, in which the same research question is evaluated by using a different experimental procedure", i.e., adhering to the same definition as Schmidt [44]. Cartwright distinguishes between the same principal categories, referring to *replications* as repeating an experiment, closely following the experimental procedures of the original ones, and refers to *reproductions* when reexamining the results from a previous experiment, using a different experimental protocol [5, 26]. We adhere to Cartwright's terminology and study one replication and one reproduction in this paper.

Despite clear definitions, there may still be room for interpretation of differences between experiments. For example, in human-oriented experiments[51], the same subjects may not be used twice due to learning effects. However, Miller [36] argues that the variation as such may be an opportunity for learning, for example, to assess the robustness of the findings, a position supported by Lindsay and Ehrenberg from a statistical point of view [33]. Shull *et al.* [46] on the other hand argue that conceptual replications are too risky to perform, as the outcome is hard to predict. Kitchenham in turn, in a rebuttal to Shull

et al., warns about replications that are too close, as they are too dependent on the instrumentation material [30].

Juristo *et al.* [24] present a series of exact–independent replications of an experiment on testing and inspection, executed over four sites, which vary along several factors: constrained vs. unconstrained time, computer availability, subjects being familiar with the technique under study or not, sequential or interleaved training, and length of the session. Unfortunately, they do not analyze these factors' influence on the outcome of the experiments. Basili *et al.* [3] discuss the experiences of conducting a series (denoted family) of experiments, which were gradually changed over replications. Differences were only analyzed qualitatively in their analysis.

2.2 Reporting practices for replication

One key precondition for conducting replications is the transparent presentation of experimental studies. The transparency should include both the experimental procedures, the data, and the statistical analyses. Miller [36] notes that reporting guidelines have been available for decades, for example, by Hoaglin and Andrews [15], as well as specific guidelines tailored to the field of software engineering [17], which are even empirically evaluated [31]. Despite these guidelines, the reporting practices are not sufficient to allow replications and proper analysis of differences between replications.

In order to replicate experiments, experiment packages [3] may help other groups of researchers to gain the information needed. Such packages should "collect information on an experiment such as the experimental design, the artifacts and processes used in the experiment, the methods used during the experimental analysis, and the motivation behind the key design decisions" [3]. While several examples of experiment packages are made available, it is still an issue of how experimenters should be acknowledged for their contribution, and how much effort they should spend explaining and answering questions on their package. Sjøberg proposed a scheme for handling these issues [48], but unfortunately it has not had any significant effect yet on research practices.

2.3 Experiment on Inspection vs. Testing

The technical field of study in this analysis are formal experiments, comparing code inspections with unit testing. The first published experiment goes back to 1972 with an early experiment by Hetzel [14]. Another classic experiment was performed by Basili and Shelby in 1987 [2]. More recent work includes a study by So *et al.*[49] in 2002, a study by Runeson and Andrews [41] in 2003, and a series of experiments presented by Juristo *et al.* [24], who also have synthesized earlier experimental findings [22, 23]. The results of the various studies are somewhat contradictory and show that this question does not yet have a clear answer. An experiment by Laitenberger [32] studied the effects of a method combining code inspection with structural testing. Laitenberger concluded that

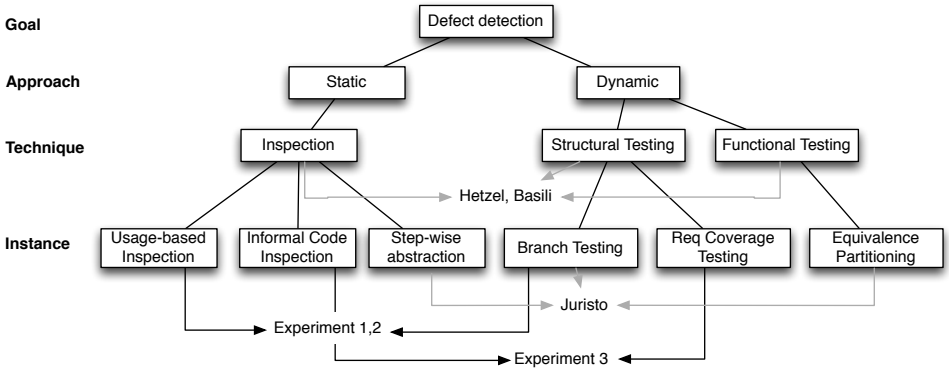

Figure 1: Classification scheme for defect detection methods

the gains of combining both were limited. A summary of findings was presented in a literature survey by Runeson *et al.* [40].

When comparing the outcomes of these experiments, it is not always clear at which level of abstraction they are compared, on a conceptual level or at the level of instances of defect detection techniques. Figure 1 presents a classification scheme to aid the comparison. All methods have the *goal of detecting defects* (the additional goal of peer-to-peer communication in inspections are never mentioned in these experiments). Defect detection methods may take a *static approach* (no execution) or a *dynamic approach* (test execution). In several of the experiments listed above, the object of study is defined as (code) *inspection* versus *structural* and *functional* (unit) *testing* [14, 2]. However, there are many possible instantiations of each of the techniques, and it is an issue of generalization across studies whether the differences observed in experiments are valid for techniques or only for instances of techniques. Juristo, for example, entitle their study a comparison between *equivalence partitioning, branch testing* and *code reading by stepwise abstraction* [24]. In this paper, we compare two different sets of instances of inspection and structural test techniques to explore the similarities and differences across these instances and thereby discuss generalization across instances of techniques.

3 Experiment Planning

The experiment planning for all three experiments, used in this replication analysis, is based on Runeson and Andrews' original experiment in 2003 [41], who report the first experiment in detail. Runeson and Andrews conducted both the original and the replicated experiment, while they handed over the experiment package to Grönblom and Porres for adaptation to the reproduction of the experiment. We summarize the experiment data and procedures here for all three

experiments.

Planning the experiment involves [51] definition of the experiment design, the treatments, defining independent and dependent variables (and how to measure them), formally stating the hypotheses, conducting a pilot study, selecting the subjects, and creating the experiment materials. Both before and after the study, threats to validity must be considered.

3.1 Goals

The goal of the study was defined as follows for experiments 1 and 2:

Analyze defect detection and localization using usage-based code inspection and structural unit testing

for the purpose of evaluation with respect to their efficiency and effectiveness for different types of defects

from the point of view of the researcher

in the context of the students at a testing course at Washington State University, verifying *C* programs from the personal software process (PSP) course [16].

For the third experiment, the goal was modified; the experiment material, the experimental context, as well as the instantiations of methods were changed, resulting in a goal definition:

Analyze defect detection and localization using requirements-based code inspection and requirements-based structural testing

for the purpose of evaluation and obtaining baseline results

from the point of view of the researcher

in the context of undergraduate and graduate students interning as programmers, working in research projects at Åbo Akademi university, verifying a *C* program and a Java Card program, which are related to embedded systems.

3.2 Definition of terms

In both experiments, the following definitions were used:

- *Defect* – umbrella term for faults and failures
- *Fault* – wrong or missing logic in program code
- *Failure* – manifestation of a fault during the execution
- *Detection* – observation of a failure

- *Localization* – finding of the underlying fault causing a failure
- *Effectiveness* – fraction of the existing defects found
- *Efficiency* – rate at which defects are found

Further, we define a *task* to be the specific combination of *technique* (test or inspection) and *program* given to a subject. We also refer to the experimental materials, i.e. programs and data collection sheets used in the experiment, as *instruments*.

3.3 Design

The experiment design involves two factors (the defect detection method and the program to be inspected) with two treatments each. In experiments 1 and 2, a two factor blocked design was used, in which subjects were randomly assigned to either defect detection technique applied to one program (*Counter*), and later applying the other technique to the other program (*Correlation*). In experiment 3, subjects first applied one randomly chosen defect detection technique to one randomly chosen program (*Acquisition* or *Channels*) and in the later session they applied the other method to the other program. All three experiments were consequently cross-over experiments, where the subjects applied one method to one program, and the other method to the other program. This can be an issue in experimentation [28] but is not discussed in this context.

3.4 Treatments

These experiments focussed on code inspection and structural unit testing, i.e. one static and one dynamic defect detection technique. However, there exist several instantiations of each technique, as discussed in Section 2.3. This involves a variation factor; however, it is not clear cut at which level of detail a 'technique' should be defined and assessed. Table 1 summarizes the factors in the three experiments, when defined at different levels of abstraction. On the approach and technique levels, they are the same for all three experiments, while the instantiation differs in experiment 3. Hence we refer to experiment 2 as a replication and experiment 3 as a reproduction.

All three experiments compare static and dynamic defect detection techniques. In experiments 1 and 2, usage-based inspection and branch coverage testing were used. Usage-based inspections are performed using usage scenarios as a guide. A set of scenarios is provided in the specification of the program. The reviewers specify the scenarios in terms of input data and expected output data. Structural unit testing was performed with the aim to reach branch coverage. The set of usage scenarios were available to the testers as well. They specify test cases in terms of the data needed and the expected output and then execute them on their assigned lab computers. Both tasks had been taught and practiced via homework before (on different programs, of course) to ensure that students follow the proper procedures.

Table 1: Experimental defect detection methods and their instantiations in the three experiments

Factor	Experiments 1 and 2	Experiment 3
Approach	Static and dynamic defect detection techniques	
Technique	Code inspection and structural unit testing	
Instantiation	Usage-based inspection and branch testing	Informal code inspection and requirements coverage testing
Coverage criterion	Usage scenarios and branch coverage	Requirements coverage

In experiment 3, defect detection techniques are structural unit testing and an informal variant of manual code inspection. The inspection procedure was informal, and therefore closer to a code review. The reviewers were not assisted by inspection aids, like checklists or lists of use cases, as suggested for some types of formal inspections, but had access to a detailed specification for the programs.

The coverage criterion used for both inspections and testing in experiment 3 was an informal requirements coverage criterion. Subjects were instructed to check that the programs conformed to the requirements, which were rather detailed and close to the code level. By contrast, experiments 1 and 2 used usage-based inspection and branch coverage as the unit testing criterion.

In summary, on the approach and technique levels, all experiments use the same methods, while the instantiation and coverage criterion used differ. The consequences of these differences are analyzed in Section 5 and discussed in Section 6.

3.5 Instrumentation

The instrumentation in the three experiments consists of task instructions, programs with a controlled set of faults, a defect log template for recording detected defects, makefiles to compile and run tests, test source files, offline and online documentation describing the behavior of the programs and forms for recording detected defects. The data collection forms are the same in the three experiments, while the programs differ, as well as the task instructions, as a consequence of the variation discussed in Section 3.4.

The first two experiments used two *C* programs providing solutions to programming assignments 3A (*Counter*) and 7A (*Correlation*) from Humprey's personal software process (PSP) course [16]. The *Counter* program is a basic line counter for *C* programs, and the *Correlation* program calculates the Pearson correlation between two sets of numerical data.

The first program in the third experiment, *Acquisition*, consists of three

Table 2: Defect and size data

Exp. 1 & 2	**Counter**	**Correlation**
Faults	9	9
LOC	190	208
# functions	7	10
Exp. 3	**Acquisition**	**Channels**
Faults	9	9
LOC	254	251
# functions	3	–
# methods	–	12

functions dealing with data acquisition functionality for spacecraft. It is implemented in *C*. The second program is a sample program taken from version 2.2.1 of Sun's Java Card Development Kit. Java Card is a minimized Java runtime environment and virtual machine designed for the hardware of smart cards. The Java Card system will be referred to as *Channels*. Data about the size of the programs can be found in Table 2.

The faults in the programs used in experiments 1 and 2 were naturally occurring during their construction by a programmer, while the faults in experiment 3 were seeded manually. The programs had no syntax errors as those could easily be found by simply compiling the programs. The faults in the programs are listed in Tables 16 and 17 in the Appendix with a classification of the type and severity, performed by the experimenters. Due to the non-complex character of the programs and faults, there is a one-to-one mapping between faults and failures.

3.6 Variables

The experiments viewed separately have two *independent* variables, the defect detection technique and the program in which to find defects, see Table 3, while in the joint analysis, the experiment itself becomes a third variable. Experiments 1 and 2 used *blocking* variables to represent skills and experience, while experiment 3 used no blocking, but random assignment. The *dependent* variables of the experiment are *time* spent on the tasks and number of defects *detected* and *localized*, as defined in Section 3.2. We also calculate the *rate*, i.e. number of defects detected and localized per time unit.

3.7 Hypotheses

The hypotheses related to the dependent variables NR and $RATE$ are listed in Table 4. We analyze the experiments firstly in Section 4 as a view for what a researcher might have done, if they had conducted each experiment in isolation. Then we analyze the replication view separately in Section 5. The analysis

Table 3: Variables used in the experiment

Name	Values	Description
Independent variables		
TECH	{Inspection, Testing}	Two techniques are applied by each subject: Inspection and Testing, as defined in Section 3.4.
PROG	{Counter, Correlation, Acquisition, Channels}	Two PSP programs in *C* for experiments 1 and 2, and two embedded systems programs for experiment 3: one in *C* and one in Java.
Blocking variables		
EXP	Ordinal	Subjects' experience with inspection and testing, measured on a four-level ordinal scale (experiments 1 and 2).
SKILL	Ordinal	Subjects' skills in inspection and testing are measured using the grading of two homework tasks (experiments 1 and 2).
Dependent variables		
TIME	Integer	Minutes spent by each subject on the task.
NR_D, NR_L	Integer	Number of defects detected and localized by each subject.
$RATE_D$, $RATE_L$	$60 * NR_D/TIME$, $60 * NR_L/TIME$	Defect's detection and localization rate (efficiency). Measured in number of defects per hour.

of the hypotheses regarding rate and number of defects are performed using the Wilcoxon signed-rank test [47]. The test was used since the data was not normally distributed, according to the Anderson-Darling test in the R `nortest` package [1]. We use a significance level of 0.05 for considering a result significant.

3.8 Pilot Study

Experiment 1 had a pilot study to evaluate the experimental instruments (six graduate students), and experiment 2 reused the same design and instruments. Experiment 3 only conducted a very limited pilot study with two subjects for inspection of the programs, but none for unit testing. Only minor changes to the instruments were added for clarification, based on the pilot studies.

Table 4: Experimental hypotheses. D=Detected, L=Localized, Insp=Inspection, Test=Testing

Null hypotheses					**Alternative hypotheses**				
$H_{NR-D,0}$	:	$NR_{D,Insp}$	=	$NR_{D,Test}$	$H_{NR-D,a}$	:	$NR_{D,Insp}$	$\neq$	$NR_{D,Test}$
$H_{RATE-D,0}$	:	$RATE_{D,Insp}$	=	$RATE_{D,Test}$	$H_{RATE-D,a}$	:	$RATE_{D,Insp}$	$\neq$	$RATE_{D,Test}$
$H_{NR-L,0}$	:	$NR_{L,Insp}$	=	$NR_{L,Test}$	$H_{NR-L,a}$	:	$NR_{L,Insp}$	$\neq$	$NR_{L,Test}$
$H_{RATE-L,0}$	:	$RATE_{L,Insp}$	=	$RATE_{L,Test}$	$H_{RATE-L,a}$	:	$RATE_{L,Insp}$	$\neq$	$RATE_{L,Test}$

Table 5: Subjects in the three experiments

	Experiment 1	**Experiment 2**	**Experiment 3**
Number of subjects	33	18	12+12
After dropouts	30	16	22
After outlier removal	30	14	22
Type of subjects	Undergraduate students		Student interns
Place	Washington State University		Åbo Akademi
Time	Spring 2003	Spring 2005	Summer 2008

3.9 Subjects

Experiment 1 had 33 undergraduate subjects, and experiment 2 had 18. These subjects were taking a senior software testing course at Washington State University. The original experiment was conducted in the Spring 2003 course, and the replication in the Spring 2005 course. The students were taught methods for testing and inspection in general; the experiment was a voluntary part of the course. Participation in the experiment was credited as 20% of the total credit of the course. For those individuals not willing to participate in the experiment, alternative assignments were available. In practice, all students volunteered for the experiment, although two students dropped out before the first session of experiment 1 and one before the second session, resulting in 30 subjects in experiment 1. Two students dropped out after the first experimental occasion in experiment 2, resulting in 16 subjects.

Experiment 3 initially included 12 subjects interning as programmers during the summer months of 2008. When they had finished their work in the project, 12 more subjects were recruited by sending out an e-mail asking for volunteers. This e-mail was sent mostly to students who were employed by the Faculty of Technology at Åbo Akademi for various research projects. Two subjects dropped out because of personal schedule changes, giving a total sample size of 22 subjects for experiment 3. The data are summarized in Table 5.

3.10 Threats to Validity

Below we summarize the key threats to each of *conclusion, internal, construct* and *external* validity for the individual experiments.

Threats to *conclusion validity* include the self reporting of time and defects data. Further, the interpretation of reported defects and removal of false positives involves a subjective component. However, as the interpretation is conducted by the same person for both treatments, there is no bias towards one or the other techniques. A major threat is the reliability of the treatment implementation, i.e. to what extent the subjects actually implemented the defined technique, or some general instance of inspection versus testing. This is an additional dimension to the approach/technique/instance issue discussed in Section 2.3 and is particularly valid when conducting experiments with student subjects. To avoid random heterogeneity between the experimental groups, experiments 1 and 2 applied blocking based on homework assignments. There is a risk that the homeworks do not properly reflect the skill levels, but this was the best available measure. Statistical tests used are rather robust, although the small number of subjects, especially in experiment 2, pose a validity threat to the statistical power.

Threats to *internal validity* include maturation and selection issues. Maturation refers to impact from subjects learning during the experiment, and thus performing better in the second round. Selection is related to how subjects are elected to participate in the experiment, and how they are assigned to different tasks. In experiments 1 and 2, one specific program (*Counter*) was used first, and then the other (*Correlation*), while in experiment 3, the order of the programs (*Acquisition, Channels*) was randomly assigned too. Randomization reduces risk, but on the other hand adds one additional degree of freedom. In all three experiments, the order of techniques was randomly assigned to subject groups. The drop-out rate may be a threat to internal validity, but in these cases it is reasonably low (see Table 5).

Threats to *construct validity* involves confounding factors that influence what is manipulated and/or observed in the experiment, but are not related to the underlying theory. There are several threats in the design of the experiment, the programs, the faults, the instructions, etc. that may have influenced the final outcome. Only differentiated replications may give an answer to the size of the risk. Further, the time factor is possibly at play in a more complex manner than assumed during experiment design.

Threats to *external validity* are primarily due to the small scale of the experiment as well to the limited experience and skill of the student subjects. Presumably, this should be a cost paid to achieve homogeneity in the subject group to increase internal validity, although the variations within the student groups were rather high as well.

3.11 Experiment Operation

The experiments were designed for time slots of two hours. The subjects were told to report for a two-hour experiment. In experiment 1, they could start any time between 11am and 1pm within the overall time frame from 11pm to 3pm. For experiment 2, the entry time was 6pm to 7pm within the overall time frame of 6pm to 9pm. However, the experimenters did not monitor the subjects' time in detail, so subjects spent between two and three hours on their task. The data collection sheet was designed to handle the situation, asking the subjects to report the exact time when they found each defect. However, as this data was entered manually by the subjects, the information may not have been consistently provided.

All testing tasks were conducted using a computer in a lab. Testers in experiments 1 and 2 were using Windows computers at Washington State University computer laboratories, and in experiment 3, Linux workstations in an Åbo Akademi computer laboratory. Programs were available electronically as archive files that contained the requirements, the source files to be checked, test source files, makefiles (facilitating automatic running of tests), and pointers where to find further information about the systems. The *Channels* program's tests were implemented using version 4 of the JUnit testing framework. The tests in *C* were implemented using a very simple ad hoc testing framework constructed by the experimenters.

The inspection tasks in experiments 1 and 2 were conducted on paper, while subjects in experiment 3 could choose whether to print out the material or review it on the screen. Defects, localized causes of defects, and time were reported by the subjects on reporting sheets. The reported defects and localized causes were then mapped by the experimenters to the list of existing defects, i.e. removing false positives. This procedure ensured consistent treatment of all reported defects.

4 Experimental Results

The analysis of the original experiment is reported by Runeson and Andrews [41], while we here analyze the three experiments one by one. Analysis of the two later experiments as replications and reproductions of the first one(s) is reported in Section 5. The purpose of showing the statistical analysis from both an individual experiment point of view and the replication point of view, is to give a view of how an independent researcher might observe the data, as opposed to a researcher with the raw data for all three experiments.

The data analysis was performed using version 2.7.2 of the open source R statistics environment. Before the analysis, the data was checked for outliers, see Section 4.1. Analyses regarding the effectiveness and efficiency of the defect detection methods are presented in Section 4.2.

4.1 Data Purification

The defect data from each of the subjects were mapped to the list of known defects in the code. Based on the description of the defect, we assessed whether the subject *detected* it or also *localized* the underlying fault. When students reported something as a defect that was not in fact a defect, we removed it as a false positive. As the subjects had nothing to lose from reporting many suspected defects, the number of false positives was in the same order of magnitude as the true positives. However, these were to a large extent phrased as vague guesses on something being a defect, rather than distinct localization of something erroneous.

The data was checked for outliers regarding time consumption. Due to the rolling scheduling of the experiment, it was not controlled that the subjects actually quit after two hours, hence there are a few subjects who spent more time. The three subjects in experiment 1 that spent 180, 141 and 138 minutes on the tasks respectively were candidate outliers, but since the time was higher for both techniques, we decided to keep the data. Five subjects spent 130 minutes, which was considered within the specified time range and since they were evenly distributed over the two groups, it was not considered a big threat. After drop-outs, we ended up with 30 subjects in experiment 1 as reported in Table 5.

In experiment 2, a few subjects also spent over two hours, due to the rolling scheduling scheme. We considered removing those spending 180 minutes as outliers, but as it concerned only two subjects, one in each group, we decided to include these subjects, regardless of time spent. Two subjects in the experiment 2 did not find a single defect in either testing nor inspection. It was clear from their materials that they misunderstood the task; hence, they were excluded from the analysis. After drop-outs and outlier removal, we ended up with 14 subjects in experiment 2.

In experiment 3, there are no outliers when it comes to time spent on the experiment which is not surprising given the enforced time limit of two hours. The closest to being considered an outlier with regards to time was one subject who according to his log template only spent 95 minutes on the task, but we decided to be inclusive. Interestingly enough, this subject also localized the largest number of faults. As there were no drop-outs, experiment 3 had 22 subjects.

4.2 Outcomes

The descriptive statistics for the efficiency ($RATE$) and effectiveness (NR) of the subjects are shown in Tables 6 and 7. The data is grouped by technique used in Table 6 and by program (number of defects only) in Table 7. Box plots of the same data can be found in Figures 2 and 3. Both sets of programs have a total of 9 seeded faults each and thus we can compare the number of defects detected directly to each other. It is worth noting that the mean number of defects found is around 30% of the total number of defects for experiments 1

Table 6: Descriptive statistics for number, rate of defects detected and localized, and time, in experiments 1-3, grouped by technique

		Mean		**Median**		**Std. Deviation**	
Exp.	**Variable**	**Insp**	**Test**	**Insp**	**Test**	**Insp**	**Test**
1	NR_D	2.43	3.27	2.00	3.00	1.52	1.78
1	NR_L	2.20	2.63	2.00	3.00	1.32	1.79
1	$RATE_D$	1.46	1.74	1.20	1.65	1.01	0.99
1	$RATE_L$	1.33	1.38	1.09	1.5	0.91	0.98
1	Time	104.7	114.8	106.0	115.5	17.47	17.95
2	NR_D	2.93	3.00	3.00	3.00	1.27	1.52
2	NR_L	2.64	2.79	3.00	3.00	1.55	1.53
2	$RATE_D$	1.94	1.20	1.76	1.12	0.93	0.62
2	$RATE_L$	1.75	1.12	1.72	1.06	1.12	0.64
2	Time	92.9	151.8	95.5	157.5	14.9	21.5
3	NR_D	2.46	0.82	2.00	0.50	1.82	1.05
3	NR_L	2.32	0.77	2.00	0.00	1.84	1.07
3	$RATE_D$	1.31	0.41	1.03	0.25	1.02	0.54
3	$RATE_L$	1.24	0.39	1.00	0.00	1.03	0.54
3	Time	115.2	117.0	120.0	120.0	7.69	5.82

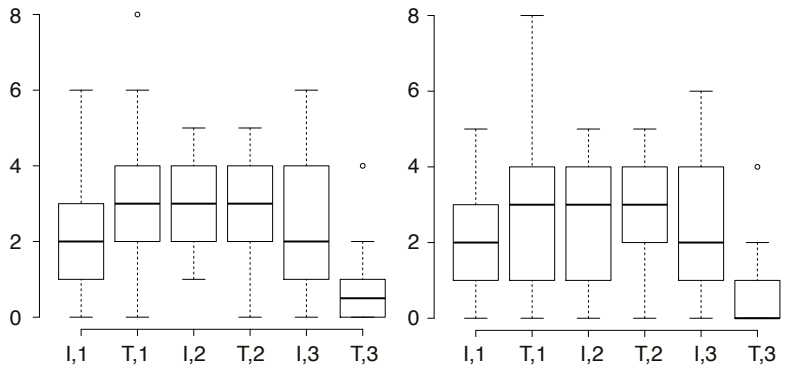

Figure 2: Boxplots of variable NR, i.e. number of detected (left) and localized (right) defects per technique for the three experiments. T=Test, I=Inspection, 1, 2, 3=Experiment 1, 2, 3

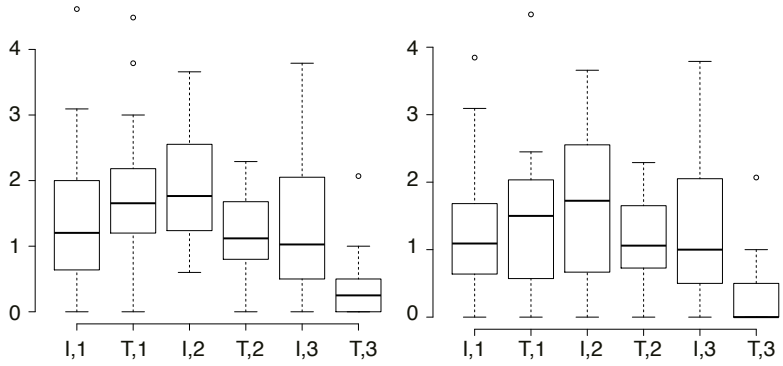

Figure 3: Boxplots of variable $RATE$, i.e. rate of detected (left) and localized (right) defects per hour per technique for the three experiments. T=Test, I=Inspection, 1, 2, 3=Experiment 1, 2, 3

and 2, and 15-20% for experiment 3, i.e. a small share of the defects. Further, the standard deviation is rather high, indicating large variations in performance within each experiment.

The experiment has two factors, paired measurements, a sample size less than 30 and data which is not normally distributed. The Wilcoxon signed-rank test is an appropriate test to compare the observed medians, both between the two programs and the two defect detection methods. In experiment 1, the testing detected more defects than inspection ($p = 0.040$), while the other differences were not significant. In experiment 2, the inspection rate was significantly higher than the testing rate for detected defects ($p = 0.031$). Note that testers spent significantly more time than inspectors in this experiment (Table 8 and Figure4). In experiment 3, p-values indicate all null hypotheses should be rejected, i.e., there is a statistically significant difference between testing and inspection, for number of defects detected, detection rate, number of defects localized, and localization rate; inspection is the better technique in experiment 3.

Analyzing the stated hypotheses only, one might conclude that experiment 1 shows significantly different effectiveness in detection of defects, experiment 2 shows significantly different efficiency in detection, and experiment 3 shows significantly higher values for inspection over testing. However, when studying the time component, notice that in experiment 1, inspectors used significantly less time than testers (104.7 vs. 114.8 minutes), $p = 0.040$. In experiment 2, the difference was even larger (92.9 vs. 151.8 minutes), $p < 0.001$. In experiment 3, the time difference is negligible (115.2 vs. 117.0 minutes), $p = 0.513$. This situation is also clearly evident from box plots of the time data, see Figure 4. We will analyze the time issue specifically in the next section.

Table 7: Descriptive statistics for number of defects detected and localized grouped by program, in addition to time. In this Table, CN = Counter, CO = Correlation, AC = Acquisition, and CH = Channels

		Mean		**Median**		**Std. Deviation**	
Exp.	**Variable**	**CN**	**CO**	**CN**	**CO**	**CN**	**CO**
1	NR_D	2.70	3.00	2.50	3.00	1.72	1.69
1	NR_L	2.07	2.76	2.00	3.00	1.60	1.49
1	Time	109.23	110.17	115.00	108.5	18.24	18.61
2	NR_D	3.14	2.79	3.00	3.00	1.41	1.37
2	NR_L	2.78	2.64	3.00	3.00	1.63	1.45
2	Time	126.86	117.79	137.00	108.5	37.83	32.83
Exp.	**Variable**	**AC**	**CH**	**AC**	**CH**	**AC**	**CH**
3	NR_D	1.87	1.41	1.00	1.00	1.67	1.71
3	NR_L	1.73	1.36	1.00	1.00	1.67	1.71
3	Time	116.27	115.95	120.00	120.00	7.23	6.51

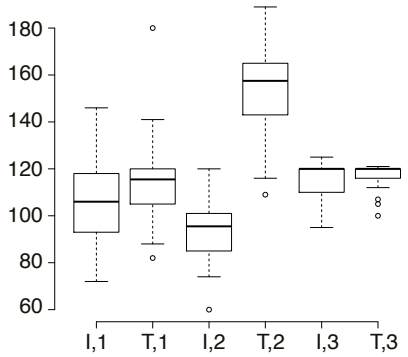

Figure 4: Boxplots over time spent per technique for the three experiments. T=Test, I=Inspection, 1,2,3=Experiment 1, 2, 3

Table 8: Summary of p-values for the null hypotheses defined in Table 4, using the two-sided Wilcoxon signed-rank test for experiments 1–3 $* = p < 0.05, ** = p < 0.01, *** = p < 0.001$

Exp.	$H_{NR-D,0}$	$H_{RATE-D,0}$	$H_{NR-L,0}$	$H_{RATE-L,0}$	$Time$
1	0.040*	0.167	0.271	0.604	0.040*
2	0.850	0.031*	0.815	0.129	<0.001***
3	0.001**	<0.001***	0.003**	0.001**	0.513

5 Replication Analysis Results

In order to better analyze the way in which our results did or did not replicate across experiments, we conducted a separate, cross-experiment analysis. Recall that from the design of the replication (experiment 2) and the reproduction (experiment 3) we expect experiments 1 and 2 to give similar results, while experiment 3 is more likely to differ. However, as shown in Section 4.2, there are differences across all the three experiments. Here we study them in more detail.

5.1 Variables and Tests

In order to conduct these analyses, we coded our study with two independent variables: i) experiment (1, 2, or 3), and ii) technique (testing vs. inspection), and five dependent variables: i) NR_D, ii) NR_L, iii) $RATE_D$, iv) $RATE_L$, and v) $TIME$. In order to determine whether to use parametric or nonparametric statistical techniques to conduct the analysis, we first conducted Shapiro–Wilk tests to test for normality of the data, all of which were significant. This implies that the data deviated somewhat from normality. However, histograms revealed that the data was simply skewed, but otherwise looked like a typical bell curve. As a cautionary measure, we conducted tests both with parametric (see Table 9) and non-parametric tests (see Table 10), noticing that they returned basically the same answer. Since typical parametric ANOVA tests are widely known to be robust to changes in normality similar to that in the data analyzed, and provided approximately the same answer as the non-parametric tests, we primarily refer to the analyses with two-factor ANOVAs. An advantage with ANOVA is that it provides analyses of the effects one by one, as well as interactions between them, i.e. it detects if there is a pairwise effect of two factors which is not the same as the sum of the effect of them taken one by one. The Tukey test is then used to identify which factors and interactions are different.

5.2 Outcomes

Table 9 shows the overall two-factor ANOVA results for the three experiments, including the five dependent variables. Before we begin, we remind the reader that in inferential statistics using two-factor ANOVAs, it is important to recall

Table 9: Summary table of the two-factor ANOVA analysis of factors Experiment and Technique, and their Interaction

	Experiment				**Technique**				**Interaction**			
	F	df	p	η_p^2	F	df	p	η_p^2	F	df	p	η_p^2
NR_D	9.585	2	<0.001	0.132	0.315	1	0.576	0.002	8.143	2	<0.001	0.114
NR_L	6.103	2	0.003	0.088	1.149	1	0.286	0.009	5.565	2	0.005	0.081
$RATE_D$	9.567	2	<0.001	0.132	4.356	1	0.039	0.033	6.299	2	0.002	0.091
$RATE_L$	5.976	2	0.003	0.087	6.283	1	0.013	0.048	3.505	2	0.033	0.053
TIME	6.904	2	0.001	0.099	44.632	1	<0.001	0.262	33.794	2	<0.001	0.349

Table 10: Summary table of the analysis using separate Kruskall-Wallis rank sum tests. Note that Kruskall-Wallis is a one-way non-parametric test, so no interactions or effect size values are provided

	Experiment			**Technique**		
	χ^2	df	p	χ^2	df	p
NR_D	17.710	2	<0.001	0.386	1	0.5344
NR_L	12.435	2	0.002	1.343	1	0.2464
$RATE_D$	19.693	2	<0.001	3.536	1	0.060
$RATE_L$	13.480	2	0.001	5.557	1	0.018
TIME	7.3541	2	0.0253	20.189	1	<0.001

that interaction effects must be considered before main effects. The reason is that strong interactions between levels of factors can strongly influence whether an ANOVA detects a significant main effect (even if spurious). As such, we consider the interactions first.

In addition to the ANOVA results, we also report a variance–accounted–for measure known as partial–eta squared (η_p^2). Partial–eta scores represent the variance accounted for by the samples and can be interpreted, if multiplied by 100, as a reasonable estimate of the percentage of the variance in the sample that is accounted for by a particular main effect or interaction. Partial–eta values are called partial because, for any particular effect, other main and interaction effects are factored out of the score. For example, a partial–eta score for a specific interaction effect represents the variance accounted for by only that specific interaction, with the variance from the main effects removed.

As can be seen from Table 9, the interactions for all five dependent variables were statistically significant, meaning that the influence of the technique variable was not entirely uniform across the experiments. This means that the techniques had different effects in at least one of the experiments. In all measures except time, the partial–eta scores (η_p^2) are generally rather low ($\eta_p^2 = .132$ or less), which means that the interaction effects between experiment and technique are small. For $TIME$, however, the interaction effect accounted for nearly 35% of

the variance in the sample. This means that the time varied across experiment (1, 2, or 3) and technique (testing or inspection). While the instructions on time behavior in experiments 1 and 2 were the same, the subjects basically chose to spend different amounts of time between the first two studies. For experiment 3, participants generally stopped after the 2 hour limit, which was not the case in experiments 1 and 2.

When we analyze the main effects of the factors, we find that:

1. the main effects for *experiment* were statistically significant for all five independent variables,
2. two of the metrics for *technique* used (NR_D and NR_L) were not statistically significant, and that
3. $RATE_D$, $RATE_L$, and $TIME$ were all significant.

The NR variables measure the number of defects, while the $RATE$ variable also includes the $TIME$ component in the denominator. As for observation 2, consider that even the NR variables are not completely independent of time, since it actually takes time to find defects. While the $RATE$ variables tie this idea directly to the equation (i. e. defects over time), humans cannot find n defects in 0 seconds. Consequently, we must acknowledge that these variables are not completely independent. In the case of all three significant measures for technique, the partial–eta values indicate that, if such main effects are legitimate, the variance they account for is smaller or comparable to that of the interaction. Specifically, the main effects for $RATE_D$ accounted for only 36.2 % of what the interaction reportedly accounts for (dividing the partial-eta values .033 / .091), whereas for $RATE_L$ it was close (90.6%), as it generally was for $TIME$ as well (75.1%). Altogether, this indicates that $TIME$ is a key factor in the variation between the experiments and between the techniques, for at least one combination of experiment and technique.

The ANOVA test reveals if there *exist* significant differences, whereas another test is needed to point out *which* factors are significantly different, for which we applied the post-hoc Tukey HSD test. Tables 11–15 show the results for each metric. While Tukey is often used for more balanced data sets than the one presented in these replications, direct pairwise comparisons using a Bonferroni correction reveals approximately the same results.

The Tukey tests for NR (Tables 11 and 12) and $RATE$ (Tables 13 and 14) show that the test technique in experiment 3 behaves significantly different compared to most of the other combinations of experiment and treatment. However, compared to testing in experiment 2, the difference is not significant.

The Tukey test for $TIME$ shows that the time metric varied across treatments and experiments, which also was evident from the box plots in Table 4. Note, for example, that the inspection group in experiment 2 differed significantly from all other conditions: (experiment 3, inspection), (experiment 1, testing), (experiment 2, testing), and (experiment 3, testing), with the exception of the combination (experiment 1, inspection). Experiment 1 with inspection,

Table 11: Summary of the Tukey HSD test for NR_D

	Mean	Std. Dev.	Tukey HSD				
			Exp. 2 Insp	**Exp. 3 Insp**	**Exp. 1 Test**	**Exp. 2 Test**	**Exp. 3 Test**
Exp. 1 – Insp	2.433	1.524	0.921	>0.999	0.304	0.868	**0.004**
Exp. 2 – Insp	2.929	1.269		0.947	0.985	>0.999	**0.002**
Exp. 3 – Insp	2.455	1.819			0.428	0.907	**0.008**
Exp. 1 – Test	3.267	1.780				0.995	**<0.001**
Exp. 2 – Test	3.000	1.519					**<0.001**
Exp. 3 – Test	0.818	1.053					

Table 12: Summary of the Tukey HSD test forNR_L

	Mean	Std. Dev.	Tukey HSD				
			Exp. 2 Insp	**Exp. 3 Insp**	**Exp. 1 Test**	**Exp. 2 Test**	**Exp. 3 Test**
Exp. 1 – Insp	2.200	1.324	0.949	>0.999	0.885	0.849	**0.016**
Exp. 2 – Insp	2.643	1.550		0.990	>0.999	>0.999	**0.007**
Exp. 3 – Insp	2.318	1.836			0.978	0.949	**0.015**
Exp. 1 – Test	2.633	1.790				>0.999	**<0.001**
Exp. 2 – Test	2.786	1.528					**0.003**
Exp. 3 – Test	0.773	1.066					

Table 13: Summary of the Tukey HSD test for $RATE_D$

	Mean	Std. Dev.	Tukey HSD				
			Exp. 2 Insp	**Exp. 3 Insp**	**Exp. 1 Test**	**Exp. 2 Test**	**Exp. 3 Test**
Exp. 1 – Insp	1.457	1.010	0.569	0.992	0.836	0.953	**<0.001**
Exp. 2 – Insp	1.939	0.929		0.326	0.983	0.267	**<0.001**
Exp. 3 – Insp	1.309	1.025			0.541	0.999	**0.016**
Exp. 1 – Test	1.737	0.994				0.453	**<0.001**
Exp. 2 – Test	1.204	0.622					0.113
Exp. 3 – Test	0.412	0.536					

Table 14: Summary of the Tukey HSD test for $RATE_L$

	Mean	Std. Dev.	Tukey HSD				
			Exp. 2 Insp	**Exp. 3 Insp**	**Exp. 1 Test**	**Exp. 2 Test**	**Exp. 3 Test**
Exp. 1 – Insp	1.331	0.905	0.706	0.999	>0.999	0.979	**0.004**
Exp. 2 – Insp	1.747	1.119		0.557	0.810	0.440	**<0.001**
Exp. 3 – Insp	1.237	1.026			0.992	0.999	**0.025**
Exp. 1 – Test	1.384	0.977				0.945	**0.002**
Exp. 2 – Test	1.122	0.638					0.168
Exp. 3 – Test	0.389	0.543					

however, appears to have been right below the threshold of statistical significance. Similarly, the testing group for experiment 2 took significantly longer to complete the tasks than every other group. In summary, the time behavior in experiment 2 was different: less time was spent on inspection and more time spent on testing.

6 Discussion

This series of three experiments gives rise to two major issues related to replicated or reproduced experiments and to factors that may differ between them; 1) variation between replications, and 2) internal versus external validity.

6.1 Variation between replications

The first replication (experiment 2) was designed to be as exact as possible. The same experimental procedures and instruments as in the original experiment were used, the same experimenters led the study, it was conducted at the same university, as part of the same university course, two years later, i.e. an *internal* replication. Despite an attempt at an exact replication, the outcomes were not the same; the differences between the two treatments were smaller in the replication. The only observable difference between the two experiments is the total time spent, which is clearly visible in Section 5.1 and is explained in Section 3.11. The third experiment was conducted by a different team of researchers (i.e. *external*), using different instruments, and different instances of the defect detection methods. Thus it was designed to be different, although applying the same basic principles.

There is the question of different skill levels for testing vs. inspection influencing the outcomes. For experiments 1 and 2 the specific unit testing and inspection techniques were taught in the same course and subjects were assigned to treatments to balance skill levels between groups as demonstrated in homework on these specific testing and inspection techniques. We do not know how this was handled in experiment 3. Moreover, experiment 3 used internship students—prior background and skill levels with testing and inspection techniques may have been less controlled. Unfortunately, there is no chain of evidence that would allow us to conclude that higher skills in the specific testing techniques than the inspection techniques caused the different results in experiment 3.

This problem of unintentional differences between replications is addressed by Shull *et al.* [45]. They discuss the differences that appear due to tacit knowledge involved in the original experiment and propose that the same experimenters conduct experiments at different locations to reduce variation. Our replication analysis indicates that it is not always enough; in our case, an automated time data collection procedure would be a better solution. Juristo and Vegas discuss the value of non-exact replications [25] and argue that we may learn about relations between variables. Similarly, Mäntylä *et al.* argue for

letting a wider range of studies being considered replications [34]. One could also discuss whether the results are robust enough if they need exactly the same procedures, researchers etc. to replicate the results. Perhaps this implies the community needs a greater understanding of other sources of variation besides how a defect detection method impacts fault finding tasks (e.g., experience, type of tasks). The systematic mapping study by da Silva *et al.* [8] clearly indicates this being a systematic issue, i.e. internal replications tend to confirm results to a much larger degree than external replications.

Another perspective would be to *not* consider unintentional differences a problem but an opportunity. Significant breakthroughs in the history of science has significant components of "mistakes", like Fleming's discovery of penicillin[1]. If we apply this principle to our experiments, the observations on time behavior should set our focus on the importance of varying time for a task properly to be efficient rather than considering time a fixed factor; an observation supported by previous studies [4]. Note that the increased time for testers in experiment 2 did *not* imply more faults being found, but resulted in a lower detection rate compared to inspection.

The reproduction study (experiment 3) was intentionally different; conducted by a separate set of researchers, using other artifacts and different instances of the inspection and test methods; primarily the design of the study is the same, and in contrast to the first two experiments, a time limit was strictly adhered to. Referring to Figure 1, this experiment compared other instances of the defect detection techniques. Not surprisingly, the outcome of the experiment is different; actually, the tendency to favor the test method in experiments 1 and 2 is turned into its clear opposite. Most probably, time is a factor here too, but the influence of artifacts, methods, experimenters, and context etc. is impossible to conclude from the experiment. In fact, in order to analyze these four factors, we would need $2^4 = 16$ experiments for a full factorial design [37], which is both impractical and should be considered only when answering a research question that is of extreme importance. If all these factors must be set to specific values for an experimental result to be valid, the external validity of software engineering experiments is very limited. Further, if the performance of two principally different detection methods turn into their opposite when using another instantiation of the methods, this also indicates the methods being very sensitive to several factors.

6.2 Internal versus external validity

There is a conflict between different types of validity to be handled when designing formal experiments [51]. When aiming for high internal and conclusion validity, populations should be as homogeneous as possible, tasks small and experiment duration short, to reduce the noise from unintentional variation. However, for external validity, practitioners with varying background and skill, real-world and non-trivial tasks should be used. This is a conflict by design in

[1] http://www.nobelprize.org/nobel_prizes/medicine/laureates/1945/fleming-faq.html

software engineering experimentation.

The type of experiment under study, two or three hour tasks on a toy size program of a few hundred lines of code, are small as they stand and can be criticized as unrealistic (lacking external validity). Even under these limited conditions, however, we have documented inherent and unintentional variations that hide effects from being statistically observable. Increasing sample sizes will eventually help sort out contributing factors, but doing so can be expensive or time consuming. Instead of increasing the sample size, we may alternatively consider running a sequence of experiments, from short, easily replicable studies with high internal validity, to increasingly longer studies with higher external validity, which are inevitably harder to replicate.

As an example, Teasley *et al.* conducted two experiments on people's tendency towards designing positive versus negative tests (published in the Journal of Applied Psychology) [50]. In the first experiment, which was very short, they asked the subjects to find the algorithm behind a sequence of numbers based on 'testing', i.e. proposing which number is next in the sequence. In the second experiment, they implemented similar choices in a computer program, asking subjects to verify that the program specification was met. These experiments aimed at understanding basic principles of human behavior in software engineering, rather than specific instances of methods. The first experiment was very small, with high internal validity, while the second was more similar to common software engineering experiments. The first experiment was very 'basic' and corroborated the outcome of several similar studies in psychology, i.e. it was highly replicable. The second experiment took its starting point in the general theory and applied it to a more software-specific task. Thereby, they managed to demonstrate the applicability of the general theory to the specific software task.

In cost estimation, similar small scale experiments are conducted [21, 29]. Thanks to their simplicity, it is easier to get a large enough number of subjects with relevant experiences and skills to participate in the experiments. Researchers in the field, for example Jørgensen *et al.*, have also isolated general factors rather than assessed specific estimation techniques, such as the impact of irrelevant and misleading information [18], the role of interdependence [19], and the impact of feed-back [20] in the cost estimation. They also analyze and discuss the role of deliberately artificial design elements in an experiment in order to enable isolation of variation factors [13]. Artificiality may then become a strength instead of a weakness.

Balancing the need for replication and real-world applicability is, however, a difficult balance. On the one hand, tiny experiments might be highly unrealistic (so, potentially, are two-hour experiments). On the other hand, if an experiment never can be replicated, how can we be sure that a researcher is correct, or even (potentially) honest? The experimental methodology is more feasible for detailed studies in the small [51] while larger studies of contemporary phenomena in its context may be more suited for case studies [42], especially "when the boundaries between phenomenon and context are not clearly evident" [52]. The methods to analyze replications of case studies are analytical rather than

statistical [42], which also moves us towards qualitative research.

7 Conclusions

This paper reports three software engineering experiments, which taken in isolation do not add very much to the knowledge base. However, the three experiments analyzed together give us much more relevant knowledge. In particular, we learn from the experiments about replication of software engineering experiments, in general.

Some in the literature have argued that replications should be as exact as possible [46], while others claim that dependent replications cause researchers to "...have invested a lot of time and effort and demonstrated very little of value" [30]. da Silva *et al.* [8] observed in their systematic mapping study that internal replications confirmed original results in 86% of the cases, while external replications did confirm original results only in 26% of the studied replications. In this analysis, we have studied one *internal replication* experiment that resembled the original one as much as possible and one *external reproduction* that substituted objects, applied other instantiations of methods, and treated time differently by imposing a time limit. We observe that even experiments designed to be exact replications contain many inherent variations; in our case the time factor, which was not designed to be a factor. The third experiment, in which the studied defect detection techniques were differently instantiated showed significant differences in the opposite direction compared to the original experiment, but it is not clear which of the factors contributed to the outcome.

Specifically from the replication analysis, we have documented that the differences, especially in regards to time, are a large contributor to the variance in our samples. Further, the analysis indicates that, if there is a statistical difference between testing and inspection, for some metrics, this effect is relatively small and was overshadowed by differences both in the experiments and in the way they interacted with the tasks. This leads us to rethink the design of the experiment. The time spent on the task seems to be a much more critical factor than other experimental factors.

We also discuss whether having even smaller scale, more artificial design elements in the experiment would be beneficial, in order to isolate the factors we hypothesize are at play. Related experimentation in software cost estimation indicates this being a feasible path forward [13].

We conclude from the replication analysis that there are variation factors between replications that may or may not be under control. We pointed out several in sections 6 and 7. Finally, if relatively small changes in a design (like changing objects and limiting time) cause basically reversals of outcomes, one needs to question just how fragile and non-generalizable some experimental results really are. Basically we are still far away from making general recommendations on specific unit testing vs. code inspection techniques. Further, reducing the complexity of experimental designs, at the cost of their realism, may be a path forward to achieve a solid empirical foundation in software engineering.

Acknowledgement

We thank Sam Grönblom and Ivan Porres, Åbo Akademi university, Finland, for providing data from experiment 3. The first author conducted parts of the work during a sabbatical at North Carolina State University, USA. We thank the anonymous reviewers to help focus the manuscript and thereby significantly improve it.

References

[1] T. Anderson and D. Darling. Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes. *Ann. Math. Statist*, 23(2), 1952.

[2] V. R. Basili and R. W. Selby. Comparing the effectiveness of software testing strategies. *IEEE Transactions on Software Engineering*, 13(12):1278–1296, 1987.

[3] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473, 1999.

[4] T. Berling and P. Runeson. Evaluation of a perspective based review method applied in an industrial setting. *IEE Proceedings – Software* 150(3): 177-184, 2003.

[5] N. Cartwright. Replicability, Reproducibility, and Robustness: Comments on Harry Collins. *History of Political Economy*, 23(1):143-155, 1991.

[6] P. Clarke and R. V. O'Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447, 2012.

[7] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, Taylor & Francis Group, 1988.

[8] F. Q. B. da Silva, M. Suassuna, A. C. C. Frana, A. M. Grubb, T. B. Gouveia, C. V. F. Monteiro and I. E. dos Santos. Replication of empirical studies in software engineering research: a systematic mapping study. *Empirical Software Engineering*, online 2012, DOI: 10.1007/s10664-012-9227-7.

[9] T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48:745–755, 2006.

[10] T. Dybå, D. I. K. Sjøberg, and D. S. Cruzes. What works for whom, where, when, and why?: on the role of context in empirical software engineering. In *Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement*, pages 19–28, 2012.

[11] P. D. Ellis. *The Essential Guide to Effect Sizes*. Cambridge University Press, 2010.

[12] O. S. Gomez, N. Juristo, and S. Vegas. Replications types in experimental disciplines. In *Proceedings of the Fourth International Symposium on Empirical Software Engineering and Measurement*, 2010.

[13] J. Hannay and M. Jørgensen. The role of deliberate artificial design elements in software engineering experiments. *IEEE Transactions on Software Engineering*, 34(2):242–259, 2008.

[14] W. Hetzel. An Experimental Analysis of Program Verification Problem Solving Capabilities as they Relate to Programmer Efficiency. *Comput. Personnel*, 3(3):10–15, 1972.

[15] D. Hoaglin and D. Andrews. The reporting of computation-based results in statistics. *The American Statistician*, 29(3):112–126, 1975.

[16] W. S. Humphrey. *A discipline for software engineering*. Addison Wesley, 1995.

[17] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *Proceedings of the 4th International Symposium on Empirical Software Engineering*, pages 95–104, 2005.

[18] M. Jørgensen and S. Grimstad. The impact of irrelevant and misleading information on software development effort estimates: A randomized controlled field experiment. *IEEE Transactions on Software Engineering*, 37(5):695–707, 2011.

[19] M. Jørgensen and S. Grimstad. Software development estimation biases: The role of interdependence. *IEEE Transactions on Software Engineering*, 38(3):677–693, 2012.

[20] M. Jørgensen and T. Gruschke. The impact of lessons-learned sessions on effort estimation and uncertainty assessments. *IEEE Transactions on Software Engineering*, 35(3):368–383, 2009.

[21] M. Jørgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33:33–53, 2007.

[22] N. Juristo, A. M. Moreno, and S. Vegas. Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, 9(1-2):7–44, 2004.

[23] N. Juristo, A. M. Moreno, S. Vegas, and M. Solari. In search of what we experimentally know about unit testing. *IEEE Software*, 23:72–80, 2006.

[24] N. Juristo, S. Vegas, M. Solari, S. Abrahao, and I. Ramos. Comparing the effectiveness of equivalence partitioning, branch testing and code reading be stepwise abstraction applied by subjects. In *Proceedings Fifth IEEE International Conference on Software Testing, Verification and Validation*, pages 330–339, Montreal, Canada, 2012.

[25] N. Juristo Juzgado and S. Vegas. The role of non-exact replications in software engineering experiments. *Empirical Software Engineering*, 16(3):295–324, 2011.

[26] N. Juristo and O. S. Gomez. Replication of Software Engineering Experiments. In B. Meyer and M. Nordio, editors, *Empirical Software Engineering and Verification*, volume 7007 of *LNCS*, pages 60–88. Springer-Verlag, 2012.

[27] V. B. Kampenes, T. Dyba, J. E. Hannay, and D. I. Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11–12):1073–1086, 2007.

[28] B. A. Kitchenham, J. Fry and S. G. Linkman. The Case Against Cross-Over Designs in Software Engineering. *11th International Workshop on Software Technology and Engineering Practice (STEP 2003)*, Amsterdam, The Netherlands, pages 65–67, Sep. 2003.

[29] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering - a systematic literature review. *Information and Software Technology*, 51(1):7–15, 2009.

[30] B. A. Kitchenham. The role of replications in empirical software engineering—a word of warning. *Empirical Software Engineering*, 13:219–221, 2008.

[31] B. A. Kitchenham, H. Al-Khilidar, M. A. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering*, 13(1):97–121, 2007.

[32] O. Laitenberger. Studying the effects of code inspection and structural testing on software quality. *In Proceedings 9th International Symposium on Software Reliability Engineering*, pages 237–246, 1998.

[33] R. M. Lindsay and A. S. C. Ehrenberg. The design of replicated studies. *The American Statistician*, 47(3):217–227, 1993.

[34] M. V. Mäntylä, C. Lasseinus, and J. Vanhanen. Rethinking replication in software engineering: Can we see the forest for the trees? In C. Knutson and J. Krein, editors, *1st International Workshop on Replication in Empirical Software Engineering Research*, Cape Town, South Africa, May 2010.

[35] J. Miller. Applying meta-analytical procedures to software engineering experiments. *Journal of Systems and Software*, 54(1):29–39, 2000.

[36] J. Miller. Replicating software engineering experiments: a poisoned chalice or the holy grail. *Information and Software Technology*, 47(4):233–244, 2005.

[37] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, Inc., 5th edition, 2001.

[38] L. Pickard, B. A. Kitchenham, and P. Jones. Combining empirical results in software engineering. *Information and Software Technology*, 40(14):811–821, 1998.

[39] A. A. Porter, H. P. Siy, A. Mockus, and L. G. Votta. Understanding the sources of variation in software inspections. *ACM Trans. Softw. Eng. Methodol.*, 7(1):41–79, 1998.

[40] P. Runeson, C. Anderson, T. Thelin, A. Andrews, and T. Berling. What do we know about defect detection methods? *IEEE Software*, 23(3):82–90, 2006.

[41] P. Runeson and A. Andrews. Detection or isolation of defects? An experimental comparison of unit testing and code inspection. *14th International Symposium on Software Reliability Engineering*, pages 3–13, 2003.

[42] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering – Guidelines and Examples*. Wiley, 2012.

[43] P. Runeson, A. Stefik, A. Andrews, S. Grönblom, I. Porres, and S. Siebert. A comparative analysis of three replicated experiments comparing inspection and unit testing. In *Proceedings 2nd International Workshop on Replication in Empirical Software Engineering Research*, pages 35–42, Banff, Canada, 2011.

[44] S. Schmidt. Shall we really do it again? the powerful concept of replication is neglected in the social sciences. *Review of General Psychology*, 13(2):90–100, 2009.

[45] F. Shull, V. R. Basili, J. Carver, J. C. Maldonado, G. H. Travassos, M. Mendonca, and S. Fabbri. Replicating software engineering experiments: addressing the tacit knowledge problem. In *Proceedings of the 1st International Symposium Empirical Software Engineering*, pages 7–16, 2002.

[46] F. J. Shull, J. Carver, S. Vegas, and N. Juristo. The role of replications in empirical software engineering. *Empirical Software Engineering*, 13(2):211–218, 2008.

[47] S. Siegel and N. Castellan. *Nonparametric statistics for the behavioural sciences*. McGraw-Hill New York, 1956.

[48] D. I. K. Sjøberg. Knowledge acquisition in software engineering requires sharing of data and artifacts. In V. Basili, H. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, and R. Selby, editors, *Empirical Software Engineering Issues: Critical Assessment and Future Directions*, volume 4336 of *LNCS*, pages 77–82. Springer-Verlag, 2007.

[49] S. So, S. Cha, T. Shimeall, and Y. Kwon. An empirical evaluation of six methods to detect faults in software. *Software Testing, Verification & Reliability*, 12(3):155–171, 2002.

[50] B. E. Teasley, L. M. Leventhal, C. R. Mynatt, and D. S. Rohlman. Why software testing is sometimes ineffective: Two applied studies of positive test strategy. *Journal of Applied Psychology*, 79(1):142–155, 1994.

[51] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Springer, 2012.

[52] R. K. Yin. *Case Study Research Design and Methods*. Sage Publications, Beverly Hills, California, 4th edition, 2009.

Appendix

Table 15: Summary of the Tukey HSD test for Time

	Mean	Std. Dev.	Tukey HSD				
			Exp. 2 Insp	**Exp. 3 Insp**	**Exp. 1 Test**	**Exp. 2 Test**	**Exp. 3 Test**
Exp. 1 – Insp	104.733	17.473	0.158	0.143	0.115	**<0.001**	0.052
Exp. 2 – Insp	92.857	14.878		**<0.001**	**<0.001**	**<0.001**	**<0.001**
Exp. 3 – Insp	115.227	7.690			>0.999	**<0.001**	0.999
Exp. 1 – Test	114.767	17.946				**<0.001**	0.995
Exp. 2 – Test	151.786	21.488					**<0.001**
Exp. 3 – Test	117.000	5.823					

Defect	**Type**[a]	**Severity**[b]	**Description**
Count 1	3	A	Missing negation in condition
Count2	3	A	Or instead of and in condition
Count3	4	B	Parameters switched in function call
Count4	3	A	Function calls in comments counted
Count5	3	A	Wrong identification of function
Count6	3	B	Line with tab not considered blank
Count7	6	C	Wrong printout
Count8	5	B	String processing wrong
Count9	4	B	Scanf compared to EOF
Corr1	1	A	Missing initialization
Corr2	2	A	Functions not called
Corr3	2	A	$n-1$ degrees of freedom instead of $n-2$
Corr4	2	A	$1-p$ probability instead of $2(1-p)$
Corr5	5	C	Constant used instead of parameter
Corr6	2	A	Wrong sign in calculation
Corr7	6	C	Misspelled printout
Corr8	6	C	Wrong precision in printout
Corr9	5	B	Wrong data type

[a] 1) Initialization, 2) Computation, 3) Control, 4) Interface, 5) Data and 6) Cosmetic.
[b] A) Major misfunction, B) Minor misfunction, C) Cosmetic.

Table 16: Defects in the PSP programs; classifications based on Basili and Shelby's scheme [2]

Defect	Type[a]	Severity[b]	Description
Acq1	5	A	Possible out of bounds error in array
Acq2	3	A	Missing proposition in logical expression
Acq3	3	A	Incorrect logical operator in logical expression
Acq4	2	B	Incorrect value stored in variable
Acq5	5	B	Two values stored at swapped indexes of an array
Acq6	3	A	Incorrect expression used in test in for-loop
Acq7	2	A	Incorrect bitmask used in computation
Acq8	2	A	Missing check for overflow when incrementing variable
Acq9	4	A	Required function only called once, instead of twice
Chan1	4	A	Removed necessary method call
Chan2	3	A	Missing case in switch statement
Chan3	3	A	Incorrect value used in logical expression
Chan4	3	A	Incorrect operator in logical expression
Chan5	4	A	Incorrect exception thrown
Chan6	1	A	Incorrect initialization of constant
Chan7	2	A	Index variable not incremented correctly in expression
Chan8	2	A	Swapped return of true and false in method
Chan9	2	A	Incorrect check for overflow when incrementing variable

[a] 1) Initialization, 2) Computation, 3) Control, 4) Interface, 5) Data and 6) Cosmetic.
[b] A) Major misfunction, B) Minor misfunction, C) Cosmetic.

Table 17: Defects in the real-time programs; classifications based on Basili and Shelby's scheme [2]