Detection or Isolation of Defects?
An Experimental Comparison of Unit Testing and Code Inspection

Per Runeson
Dept. of Communication Systems, Lund University
Box 118, SE-221 00 LUND, Sweden
per.runeson@telecom.lth.se

Anneliese Andrews
School of EE&CS
Washington State University
P O Box 642752, Pullman, WA 99164, USA
aandrews@eecs.wsu.edu

Abstract

Code inspections and white-box testing have both been used for unit testing. One is a static analysis technique, the other a dynamic one, since it is based on executing test cases. Naturally, the question arises whether one is superior to the other, or, whether either technique is better suited to detect or isolate certain types of defects. We investigated this question with an experiment with a focus on detection of the defects (failures) and isolation of the underlying sources of the defects (faults). The results indicate that there exist significant differences for some of the effects of using code inspection versus testing. White-box testing is more effective, i.e. detects significantly more defects while inspection isolates the underlying source of a larger share of the defects detected. Testers spend significantly more time, hence the difference in efficiency is smaller, and is not statistically significant. The two techniques are also shown to detect and identify different defects, hence motivating the use of a combination of methods.

Keywords: Controlled Experiment, Empirical Study, Defect detection, Defect isolation, Unit testing, Code Inspection.

1. Introduction

The reliability of a software product is dependent on the defects in the software. Defects range from the small annoying misspellings in the user interface to the more disturbing defects that crash not only the current program, but also destroy data and stop other programs. The cost of preventing and removing defects is enormous (estimated at over $50 billion per year in the US alone), but the knowledge regarding efficiency and effectiveness of different methods is limited, despite the fact that defect detection is the topic of a large body of experiments in software engineering.

In this paper we focus on methods to detect defects in the code and isolate the cause of the defects. We distinguish between dynamic methods, i.e. testing [7], and static methods, i.e. inspection [4]. Dynamic methods execute the program and thereby detect deviations from the expected behavior. Methods to define what to test include coverage-based testing. Tests are defined and executed until a coverage criterion is fulfilled. Coverage criteria include code coverage (each statement is executed once), branch coverage (each branch in the execution tree is covered) and path coverage (each path in the execution tree is covered) [7]. Static methods scrutinize the printout of the code manually, without execution. Methods to guide the inspection include check-lists (a list of check items to follow), abstractions (the functionality of each part is abstracted from the code) and scenarios (sequences of events to trace though the code) [3].

We also distinguish between defect detection and defect isolation. Defect detection refers to the observation that the program behavior is wrong, i.e. the manifestation of a failure. For example, the output of the program is 3 instead of 4. Defect isolation refers to the isolation of the cause of the defect, i.e. the fault in the code. For example, the wrong parameter was passed to a function, leading to an output of 3 instead of 4.

Defect detection and isolation on the code level requires comprehending the intended and actual behavior of the program. Existing studies point to the need for systematic comprehension of the code, if one wants to either write correct programs or find all the faults [12]. In these studies, it was shown that a systematic (line-by-line) approach was superior to an opportunistic or “as needed” comprehension strategy. Knowledge of the application area is also important. Most of the existing studies of program comprehension were done using static analysis (i.e. on a paper printout of a small program) [2, 15, 23, 24]. However, there are also experiments investigating the difference between static analysis through code reading without tool support compared to using a sophisticated browsing and analysis tool capable of dataflow analysis, control-flow browsing, and switching between levels of detail (call graph versus full code) [14, 13]. Behavior between the two groups during debugging varied greatly. The group with the tool dis-
played much more “guessing around” behavior (finding answers to hypotheses was immediate and cheap), but they did not engage as much in the painstaking in-depth analysis of the subjects who had only a printout of the code in front of them. It stands to reason, that code inspection and testing also would result in different behavior and possibly different effectiveness and efficiency for certain types of defects. In summary, this experiment is motivated both by results in program comprehension as well as by existing results of inspection experiments.

In the area of defect detection, several experiments have been performed. If limiting the scope to comparisons between code inspection and unit testing, the experiments range from Hetzel’s [6] early experiment in 1976, through Basili’s and Selby’s classical experiment mid 1980’s [1], to a recently published study by So et al. [20]. The experiments contributed different and sometimes conflicting results, showing that the question of which method is the best does not have a single and simple answer. Laitenberger presents a survey over performed experiments related to his study comparing the combination of inspection and testing [11]. In the experiment code inspection was first performed, then followed by structural testing. The outcome of the Laitenberger experiment is that the gains of combining inspection and testing like this are limited.

In order to further understand the nature of the techniques, we launched an experiment, with the following purpose:

- **Analyze** defect detection and isolation 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 (WSU), verifying c programs from the personal software process (PSP) course [8].

Some of the terms used in the paper are defined below:
- **Defect** - umbrella term for faults and failures.
- **Detection** - observation of a failure.
- **Isolation** - identification of the underlying fault.
- **Effectiveness** - ratio of the existing defects found.
- **Efficiency** - the rate at which defects are found.
- **Fault** - wrong or missing function in the code.
- **Failure** - the manifestation of a fault during execution.
- **Usage scenario** - a sequence of steps describing an interaction between a user and a program.
- **Use case** - set of usage scenarios tied together by a common user goal.

The paper is structured as follows. The planning and definition of the experiment is presented in Section 2. Section 3 reports on the operation of the experiment and Section 4 shows the data analysis. The interpretation of the data is summarized in Section 5.

## 2. Experiment planning

The experiment is conducted according to the procedures proposed by Wohlin et al. [25]. The planning involves defining the treatments (Section 2.1), setting up the variables to measure (Section 2.2), formally stating the hypotheses to be tested (Section 2.3), conducting a pilot study (Section 2.4), selecting the subjects to be involved in the study (Section 2.5), choosing the experimental design (Section 2.6) and developing the instruments for the experiment (Section 2.7). Both before and after the experiment, issues of validity have to be considered (Section 2.8).

### 2.1. Treatments

The focus for this experiment is primarily the concepts of static and dynamic verification. The specific techniques are of secondary interest. However, the techniques used in the experiment are two specific instances of static and dynamic verification, namely usage-based inspection and coverage-based testing.

Usage-based inspections are performed using usage scenarios as a guide. The technique can be used for code inspections [3], design inspections [21, 22] as well as requirements inspections [5]. 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. Then the “execution” of each scenario is traced through the code using the input data. During the “execution”, defects and the cause of the defects are isolated and reported.

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. The testers execute the test cases and report defects found. If needed for further execution, they were allowed to repair the program.

### 2.2. Variables

The variables of the experiment are summarized in Table 1. This experiment has two independent variables: the defect detection technique and the program to find defects in. We control the variable of subject experience, using a pre-test questionnaire and the subject skills by the grading of two home-work tasks, one for inspection and one for testing. The control variables are used to achieve balanced groups with respect to experience and skills.

The dependent variables are related to the time spent on the tasks and the defects found. For inspections, the time spent on overview reading, definition of scenarios for the inspection and the inspection as such are separated. For testing, the units are overview, definition of test cases, test execution and repair. In the analysis, however, only the total time is used. The defects are measured based on the subjects’ defect logs. For each subject, the number of defects detected and isolated are measured. The efficiency,
i.e. the defects isolated and detected per time unit is calculated as well. We also measure how many subjects detected and/or isolated a particular defect.

2.3. Hypotheses

The basis for the hypotheses is that we assume inspection and testing perform better for different purposes. Based on the discussion in the introduction, we expect testing to be more effective and efficient in detecting defects (via failures), while inspection is more effective and efficient in isolating the source of defects (faults).

We set up six null hypotheses that there are no differences in effectiveness, efficiency (rate) and defects found between code inspection and unit testing, neither with respect to detection nor isolation of defects.

- \( H_{eff-det,0} : \text{DEF}_{D,\text{Insp}} = \text{DEF}_{D,\text{Test}} \)
- \( H_{rate-det,0} : \text{EFF}_{D,\text{Insp}} = \text{EFF}_{D,\text{Test}} \)
- \( H_{defect-det,0} : \text{FND}_{D,\text{Insp}} = \text{FND}_{D,\text{Test}} \)
- \( H_{eff-id,0} : \text{DEF}_{I,\text{Insp}} = \text{DEF}_{I,\text{Test}} \)
- \( H_{rate-id,0} : \text{EFF}_{I,\text{Insp}} = \text{EFF}_{I,\text{Test}} \)
- \( H_{defect-id,0} : \text{FND}_{I,\text{Insp}} = \text{FND}_{I,\text{Test}} \)

The alternative hypotheses for detection effectiveness and efficiency are defined in favor of testing:

- \( H_{eff-det,a} : \text{DEF}_{D,\text{Insp}} < \text{DEF}_{D,\text{Test}} \)
- \( H_{rate-det,a} : \text{EFF}_{D,\text{Insp}} < \text{EFF}_{D,\text{Test}} \)

The alternative hypotheses for isolation effectiveness and efficiency are defined in favor of inspection:

- \( H_{eff-id,a} : \text{DEF}_{I,\text{Insp}} > \text{DEF}_{I,\text{Test}} \)
- \( H_{rate-id,a} : \text{EFF}_{I,\text{Insp}} > \text{EFF}_{I,\text{Test}} \)

The alternative hypotheses for how many of the subjects found each defect are not defined for each defect, but for the whole distribution. Hence the alternative hypotheses state that the two techniques detect and isolate different defects:

- \( H_{defect-det,a} : \text{FND}_{D,\text{Insp}} \neq \text{FND}_{D,\text{Test}} \)
- \( H_{defect-id,a} : \text{FND}_{I,\text{Insp}} \neq \text{FND}_{I,\text{Test}} \)

The analysis of the hypotheses regarding effectiveness and efficiency is performed using the Wilcoxon ranked sign test [19]. This test is the non-parametric counterpart to a paired t-test and was used since the data could not be proven to have normal distribution. The hypothesis regarding fault distributions is analyzed using a \( \chi^2 \)-test (see [16, 19] for more detailed procedures).

In the tests, a significance level of 0.05 is chosen for considering the result significant, i.e. the probability for the result not to be random is 95%.

2.4. Pilot study

A pilot study was conducted to evaluate the instruments and experiment procedures. Six graduate students attended the pilot study. They used preliminary versions of the instruments with slightly different sets of defects and based on the outcome, the final versions were developed. The changes included removal of some defects and clarification of guidelines. Most of the graduate students had recently taken a graduate course on experimentation in software engineering, and were hence well skilled to give feedback on the instruments.

2.5. Subjects

The subjects of the experiment were 30 undergraduate students. The subjects were attending a senior year software testing course at Washington State University (WSU), Pullman, WA, USA. The students are taught methods for testing and inspection in general, and the experiment is a voluntary part of the course. The participation on the experiment was credited 20% of the total credit of the

Table 1. Variables used in the experiment

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent</td>
<td>TECH {Insp, Test}</td>
<td>Two techniques are applied by each subject: Inspection and Testing.</td>
</tr>
<tr>
<td>Controlled</td>
<td>PROG {Counter, Correlation}</td>
<td>The objects are two c programs from the PSP course, see Table 4</td>
</tr>
<tr>
<td>Controlled</td>
<td>EXP Ordinal</td>
<td>The experience with inspection and testing is measured on a four-level ordinal scale.</td>
</tr>
<tr>
<td>Controlled</td>
<td>SKILL Ordinal</td>
<td>The skills in inspection and testing are measured using the grading of two homework tasks.</td>
</tr>
<tr>
<td>Dependent</td>
<td>TIME Integer</td>
<td>The time spent by each subject in performing the task. The time unit used is minutes.</td>
</tr>
<tr>
<td>Dependent</td>
<td>DEF\textsubscript{I}, DEF\textsubscript{D} Integer</td>
<td>The number of defects isolated and detected by each reviewer is recorded, excluding false positives.</td>
</tr>
<tr>
<td>Dependent</td>
<td>EFF\textsubscript{I}, EFF\textsubscript{D} 60<em>DEF\textsubscript{I}/TIME, 60</em>DEF\textsubscript{D}/TIME</td>
<td>The defect isolation and detection efficiency. The unit is defects per hour.</td>
</tr>
<tr>
<td>Dependent</td>
<td>FND\textsubscript{I}, FND\textsubscript{D} Integer</td>
<td>The number of subjects which have found a certain defect in a specific program.</td>
</tr>
<tr>
<td>Dependent</td>
<td>ISOL \sum FND\textsubscript{I} \sum FND\textsubscript{D}</td>
<td>The ratio of the subjects that isolated the defect if it was detected. ISOL is defined being zero if the defect was found or isolated by none of the subjects.</td>
</tr>
</tbody>
</table>
course and for those not willing to participate in the experiment, alternative tasks were available for the corresponding part of the course credit. However, all students were willing to participate in the experiment.

Of the 30 students actually participating in the experiment, two dropped out before the first session since they dropped the course for personal reasons, and one dropped the experiment after the first session, because he misread the location where it was to be held.

The subjects’ experience in software testing and inspection are assessed in a pre-test with 10 questions, each on a four-level Likert scale ranging from “no experience” to “long industrial experience (>3 years)”. In addition, the number of courses in computer science and software engineering taken is measured. No subject had taken the personal software process (PSP) course [8]. This was important, since the instruments are taken from that course and students with prior exposure to that course would have been more familiar with those programs than those who did not take the course.

The subjects’ skills are assessed using the grading of two home-work tasks in the course, one for testing and one for inspection. The students in the two experiment groups scored similarly as depicted in Table 2. The large standard deviation regarding skills comes from some students’ failure to hand in homework on time (9 students for the testing homework and 2 students for the inspection homework), and receiving a score of zero.

Table 2. Average and standard deviation for the pre-tests per group.

<table>
<thead>
<tr>
<th>Group</th>
<th>Experience (max 52)</th>
<th>Skills (max 50)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>StDev</td>
</tr>
<tr>
<td>Inspection then Test</td>
<td>25.44</td>
<td>2.71</td>
</tr>
<tr>
<td>Test then Inspection</td>
<td>25.24</td>
<td>2.98</td>
</tr>
</tbody>
</table>

2.6. Design

The design chosen is a two-factor blocked design [10]. The independent variables, or factors are:

- Technique used (testing, inspection)
- Instrument used (program Counter, program Correlation)

The factor under investigation is the technique used. In order to investigate the confounding effects of the instruments, each group applies the two techniques on different instruments, and in order to investigate the confounding effect of history and technique, the groups apply the techniques in different order. The design is presented in Table 3, where the numbers refer to the subjects.

2.7. Instrumentation

Two C programs are used as objects for defect detection and isolation. The programs are implementations of tasks 3A and 7A from the PSP course [8], here referred to as Counter and Correlation, respectively. The defects in the programs are “natural” defects which were inserted during the development of the programs in the PSP context. The programs are taken from a baseline version prior to compilation. The compilation errors are removed before the programs are used as experiment objects. The pilot study made sure that it was feasible to detect and identify the defect sets through testing and inspection. The programs are the same as those used in an earlier inspection experiment [17], although we reduced the number of defects.

The Counter program is a simple line of code (LOC) counter for C programs. It is based on a coding standard that requires each logical line of code to be a physical line. Hence, physical lines containing any program statements are counted. Comment lines and empty lines are not counted. The Correlation program calculates the correlation between two sets of data and the statistical significance of the correlation, based on a t distribution. The distribution function is integrated numerically. Basic data about the programs are presented in Table 4.

Table 3. Two-factor blocked design.

<table>
<thead>
<tr>
<th>Program used</th>
<th>Counter</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique</td>
<td>Inspection</td>
<td>Testing</td>
</tr>
<tr>
<td>Counter</td>
<td>1-15</td>
<td>16-30</td>
</tr>
<tr>
<td>Correlation</td>
<td>16-30</td>
<td>1-15</td>
</tr>
</tbody>
</table>

Table 4. Defect and size data

<table>
<thead>
<tr>
<th>Program</th>
<th>Counter</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>LOC</td>
<td>190</td>
<td>208</td>
</tr>
<tr>
<td>McCabe Cyclomatic Complexity</td>
<td>27</td>
<td>24</td>
</tr>
<tr>
<td>#procedures</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

The defects are classified using a scheme based on Basili and Selby’s scheme [1] with defect classes for 1) Initialization, 2) Computation, 3) Control, 4) Interface, 5) Data and 6) Cosmetic. The severity of the defects is classified as A) Major misfunction, B) Minor misfunction, C) Cosmetic. The defects with corresponding classifications are presented in Table 5.

In addition to the programs, the experiment instruments comprise program specifications including requirements, usage scenarios and an example use case, process descriptions for the two techniques, a combined time and defect log template, and pre- and post-test questionnaires. The specifications are derived from Humphrey’s PSP book [8]. The process descriptions use the style of the PSP process. The log template is derived for experimental purposes and used in several earlier experiments, for example [21]. The pre-test contains 10 questions regarding experience of soft-
2.8. Validity

There are always threats to the validity of an empirical study. To enable an analysis of the validity of the current study, we present a list of identified risks, derived based on Wohlin et al. [25] and the measures taken to reduce the risks. As we investigate the basic behavior of defect detection techniques, we give priority to internal and construct validity before external validity.

Conclusion validity. These threats are concerned with the ability to draw conclusions regarding relations between the treatment and the outcome.

For the statistical analysis, we use non-parametric tests, thus not requiring any specific underlying distribution of the data. The $\chi^2$ test is also found to be robust in earlier analyses [16].

The measures used involve human judgement at one point, i.e. in the interpretation of the submitted data regarding detected and isolated defects. The interpretation is performed by one person to ensure consistent classification, although this may lead to some human bias. However, the bias is the same for both treatments. The time measures do not involve any judgement since only total time is used in the analysis.

One threat to the conclusion validity is the reliability of the treatments, i.e. do the subjects really apply usage-based inspections and coverage-based testing respectively. The homework assignment for each technique is our indicator whether they understand the techniques as such, and as 11 out of 60 homeworks were not handed in properly, this is a threat. However, as the experiment compares inspection and testing, the subject are very much bound by the working environment. Either they have access to a computer or not. Inspection subjects were not able to apply dynamic techniques, and testing subjects did execute the program, hence they applied inspection and testing respectively, although we cannot provide evidence to what extent they followed the defined procedures.

To increase the reliability of the treatment and to reduce the risk of random heterogeneity, we blocked the subjects by having a blocked design, and balanced the groups based on the pre-test survey of experiences and skills.

Internal validity. The internal validity threats are influences that may affect the independent variable without the researchers’ knowledge.

The order in which different techniques are applied may favor the one applied later as some learning effects may occur. Hence, one group applied inspection first, the other testing first. The order of the instruments may also impact. However, this was considered a smaller threat than the risk of leaking information between the groups, hence both groups used the Counter program for the first occasion and the Correlation program for the second one.

The instruments are used in other experiments in similar form, hence they are not considered being a threat to the internal validity. In addition, the pilot study sorted out a few remaining issues regarding the instruments.

The selection and mortality issues are under control. The students were free to not participate or drop out after one session. Two students dropped the course before the experiment and one student dropped after the first experiment session.

Both groups applied both techniques and were given the same amount of credits independently of the results. Hence, we do not consider any compensatory issues being a risk. The motivation for the students is the gained practical skills in the two techniques.

Construct validity. These threats concern the concept behind the experiment.

---

Table 5. Defects in the programs.

<table>
<thead>
<tr>
<th>Defect</th>
<th>Typea</th>
<th>Severityb</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count1</td>
<td>3</td>
<td>A</td>
<td>Missing negation in condition</td>
</tr>
<tr>
<td>count2</td>
<td>3</td>
<td>A</td>
<td>Or instead of and in condition</td>
</tr>
<tr>
<td>count3</td>
<td>4</td>
<td>B</td>
<td>Parameters switched in function call</td>
</tr>
<tr>
<td>count4</td>
<td>3</td>
<td>A</td>
<td>Function calls in comments counted</td>
</tr>
<tr>
<td>count5</td>
<td>3</td>
<td>A</td>
<td>Wrong identification of function</td>
</tr>
<tr>
<td>count6</td>
<td>3</td>
<td>B</td>
<td>Line with tab not considered blank</td>
</tr>
<tr>
<td>count7</td>
<td>6</td>
<td>C</td>
<td>Wrong printout</td>
</tr>
<tr>
<td>count8</td>
<td>5</td>
<td>B</td>
<td>String processing wrong</td>
</tr>
<tr>
<td>count9</td>
<td>4</td>
<td>B</td>
<td>Scanf compared to EOF</td>
</tr>
<tr>
<td>corr1</td>
<td>1</td>
<td>A</td>
<td>Missing initialization</td>
</tr>
<tr>
<td>corr2</td>
<td>2</td>
<td>A</td>
<td>Functions not called</td>
</tr>
<tr>
<td>corr3</td>
<td>2</td>
<td>A</td>
<td>n–1 degrees of freedom instead of n–2</td>
</tr>
<tr>
<td>corr4</td>
<td>2</td>
<td>A</td>
<td>1–p probability instead of 2(1–p)</td>
</tr>
<tr>
<td>corr5</td>
<td>5</td>
<td>C</td>
<td>Constant used instead of parameter</td>
</tr>
<tr>
<td>corr6</td>
<td>2</td>
<td>A</td>
<td>Wrong sign in calculation</td>
</tr>
<tr>
<td>corr7</td>
<td>6</td>
<td>C</td>
<td>Misspelled printout</td>
</tr>
<tr>
<td>corr8</td>
<td>6</td>
<td>C</td>
<td>Wrong precision in printout</td>
</tr>
<tr>
<td>corr9</td>
<td>5</td>
<td>B</td>
<td>Wrong data type</td>
</tr>
</tbody>
</table>

b. A) Major malfunction, B) Minor malfunction, C) Cosmetic.
Mono-operation threats are reduced by having two different programs in different domains (text processing and statistical analysis respectively), although of the same size and complexity. To reduce mono-method threats, the post-test questionnaire was added to enable qualitative validation of the results. The subjects were very much in favor of testing, which is in line with the overall results, but they graded the program as easiest to comprehend in which they found the least number of defects.

Other threats to construct validity are social threats, like hypothesis guessing and experimenter expectancies. As neither the subjects nor the experimenters have any interest in favor of one technique or another, we do not expect this being a large threat.

External validity. Threats to external validity are naturally that the subjects are undergraduate students and that the objects are programs from an academical environment. Using students as subject in general, and undergraduate students in particular is debated [9, 18]. The sizes of the program objects are relevant for the time frame, although the context would be more complex in a real program. Problems like violated real-time constraints and memory management issues were not present in the small programs. However, in the overall balance between validity threats, these deficits are known, and hence we make no statements about the generalizability of the results.

3. Experiment operation

The experiment was conducted during one week in February 2003. The training was performed in the weeks before and feed-back was given to the participants a few weeks later.

For each of the two techniques used, the subjects were taught at least one lecture and a home-work assignment was given on each technique. All participants did not attend the lectures nor handed in their home-works. We have no statistics on attendance of the lectures, while we know that 9 subjects did not hand in the testing homework properly and 2 subjects did not hand in the inspection homework.

Due to scheduling constraints given by other classes, the subjects attended the experiment on a rolling scheme. The experiment was conducted in classroom settings under continuous supervision of assistants. The subjects attended two hours Tuesday and two hours Thursday during the time slots defined in Table 6. In addition, one subject performed the task Wednesday-Friday instead.

Table 6. Experiment schedule

<table>
<thead>
<tr>
<th></th>
<th>Tuesday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td>11am-3pm</td>
<td>Counter, test</td>
<td>Correlation, inspection</td>
</tr>
<tr>
<td></td>
<td>Counter, inspection</td>
<td>Correlation, test</td>
</tr>
<tr>
<td>5:30pm-7:30pm</td>
<td>Counter, test</td>
<td>Correlation, inspection</td>
</tr>
</tbody>
</table>

The inspection session was performed in a classroom and used paper only. The subjects inspecting the Correlation program were asked to bring a calculator to enable calculating expected results of the scenarios. The testers performed their task in the student lab. Hence they were performing the experiment in their ordinary working environment. They could choose either a MS Visual or a Linux/GNU environment, depending on their own preferences.

Two subjects in each testing group had problems in setting up the GNU compiler to find the correct libraries. However, this problem accounted for less than 10 minutes each.

4. Data analysis

The data analysis was performed using Statview 5.01 for Windows from SAS Institute. Before the analysis, the data was checked for outliers (Section 4.1). Analyses regarding the effectiveness and efficiency are presented in Section 4.2 and regarding the distribution of defects found for different techniques is dealt with in Section 4.3. The analysis of the post-test is presented in Section 4.4.

4.1. Data purification

The data was checked for outliers regarding time consumption. Due to the rolling scheduling of the experiment, it was not checked that the subjects actually quit after two hours, hence there are a few subjects who spent more time. One subject who spent 180 minutes on one task was removed from the data, since this would unbalance the effectiveness analysis. One subject who spent 141 and 138 minutes on the tasks respectively is a candidate outlier, but since the time was higher for both techniques, and the single data point does not impact the analysis results, it was decided to keep it. Five subjects spent 130 minutes. This was considered within the specified time range and since they were evenly distributed over the two groups, it was not considered a big threat.

The pre-test data regarding experience and skills could also be a source for finding outliers. The subjects which had not handed in their homework are candidates for being treated as outliers. However, it seems from the analysis that this is more a matter of discipline than of knowledge and skills. The subjects who had not handed in their homework did not perform differently from the rest of the group. However, to be on the safe side, this is taken into account regarding the validity of treatment implementation, see Section 2.8.

The average detection rate of 33% is in the same range as in similar experiments, although at the lower end of the range. Laitenberger provides a survey of six experiments [11] where the average percentage of defects found range from 30% to 58% for different techniques.

The data was tested for normality. Since normality could not be shown, non-parametric tests are used in the analyses.
4.2. Efficiency and effectiveness

The descriptive statistics of the efficiency and effectiveness data are presented in Table 7. Median and mean values and standard deviation for the number of detected defects (DEFD), number of isolated defects (DEFI), detected defects per hour defects (EFFD), and isolated defects per hour (EFFI) are shown below. The data is grouped per technique (inspection, test) in Table 7 and per program (Counter, Correlation) in Table 8 to enable analysis of the cause of the differences. The DEFD and DEFI variables are also shown in box-plots in Figures 1 and 2 respectively.

The Wilcoxon signed rank test was performed to test the hypotheses, see Table 9. The number of defects detected is significantly larger for testing than for inspection. Hence we can reject hypothesis H\text{eff-det,0} in favor of the alternative hypotheses. The other differences are not statistically significant.

The same analyses are performed to test whether the type of program accounts for the differences. There is no statistically significant difference between the programs.

To enable analysis of the cause of the difference, the time spent for each program and technique is analyzed, see Figure 3. Testers spent on average 112.5 minutes and inspectors spent 103.8 minutes (median 115 vs. 105). The variation is larger among the inspectors; standard deviation is 13.3 minutes for testers and 17.1 minutes for inspectors.

The difference regarding the average of 8.7 minutes is statistically significant (p=0.03).

There is almost no difference between the mean time spent on the two programs (108.5 vs. 107.9 minutes) although there is a larger variation for the Counter program, which was used first (Standard deviation 18.1 vs. 13.4). This difference is also visible in the median value (115 vs. 108 minutes), i.e. the time distribution is skewed.

We have concluded that testers find significantly more defects in significantly more time. An obvious question is whether this is because testers spend more time. To some extent this is true, but it is not the full answer. As can be seen in the EFFD figures, the test group performs better although the difference is not statistically significant. A tester
detects on average 1.8 defects per hour. If they spent 10
minutes less time, the mean number of defects would
decrease from 3.38 to 3.08, which is still 0.6 defects more
than inspectors find on average. Hence, we consider the
effect is related to the difference between the techniques,
not to the time difference.

4.3. Defect distribution

The distribution of defects found by different techniques
(FND in Table 1) is analyzed using bar plots and a $\chi^2$-test,
see [16, 19]. The analyses are performed for each defect
and for defect types, both for detected and isolated defects.

The bar plot for number of subjects detecting each
defect (failure) is presented in Figure 4. It can be noted that
some defects are to a higher extent detected by test (corr1,
corr8, count3) while others are to a higher extent detected
by inspection (count5, count9). The $\chi^2$-test of the null
hypothesis that there is no difference between the profiles
gives a $\chi^2$-value of 25.8 and p-value of 0.078. The exist-
ence of a difference is hence not significant on a level of
0.05.

The corresponding bar plot for number of subjects isolat-
ing each defect (fault) is found in Figure 5. The $\chi^2$-test
gives a $\chi^2$-value of 30.3 and p-value of 0.017. The exist-
ence of a difference is hence significant on a level of 0.05.
It can be noted that for some specific defects (count3,
count5, count6) fewer testing subjects isolate the defects
compared to those detecting the defects.

The distribution analysis for the detected defects (fail-
ures), grouped by the defect type, is presented in Figure 6.
More initialization defects are found by testers, but other-

Figure 3. Boxplot of time spent per program
(left) and per technique (right).

Figure 4. Bar chart of number of subjects detecting
each defect (failure).

Figure 5. Bar chart of number of subjects isolating
each defect (fault).

Figure 6. Bar chart of number of subjects detecting each
defect (failure) grouped by defect type.
1) Initialization, 2) Computation, 3) Control,
4) Interface, 5) Data and 6) Cosmetic

Figure 7. Bar chart of number of subjects isolating each
defect (fault) grouped by defect type.
1) Initialization, 2) Computation, 3) Control,
4) Interface, 5) Data and 6) Cosmetic
wise the differences are small and not statistically significant. The corresponding bar chart for isolated defects (faults) is shown in Figure 7. These differences are not significant either.

In order to further understand the differences between inspection and test, we analyzed the isolation rate for the two techniques (ISOL in Table 1). A box plot is shown in Figure 8 showing that inspection detects a larger share of the source of the defects compared to testing. The means are 84% and 67% respectively. The difference is statistically significant (p=0.05).

We also analyzed whether the techniques behave differently for defects of different severity. The distribution for detection of defects (failures) of type A) Major misfunction, B) Minor misfunction, C) Cosmetic shows that inspection and testing detect critical defects at the same rate, while minor issues are found at a higher rate by testing, see Figure 9. The same relations hold for isolation of defects (faults), see Figure 10, but at a slightly lower level. Neither of the differences is statistically significant, due to the small number of categories and the small number of defects in each category.

4.4. Post-test

The post-test consisted of 16 questions grouped into three sections. The data is summarized in the appendix. The first group of questions dealt with the task as such. The subjects reported that the instructions were acceptable and that the tasks were non-trivial.

Regarding the two programs, most subjects found the Counter program to be easier to understand. They had similar experience in the two domains, but they thought that they performed slightly better for the Correlation program, see Table 8.

The remaining 10 questions were about favoring testing or inspection. The general opinion before the experiment was very much pro testing, and just a few subjects changed their mind during the experiment. Most subjects thought testing is better for interface issues, functional, logic and behavioral issues. To detect mismatches between design and code, the opinions were balanced between inspection and test. Most subjects considered testing best for detecting defects, while for isolation of defects, the opinions were balanced.

5. Interpretation and conclusion

Based on the statistical data analysis in Section 4 we can conclude that the testers in the experiment detect significantly more defects (failures) than inspectors do (p=0.03). At the same time, inspectors spend significantly less time than testers (p=0.03). This might lead us to the conclusion that, while more expensive with regards to time, testing is more effective. However, the data also shows that when inspectors detect defects, they isolate them more successfully than testers (p=0.05), i.e. they find the fault behind the failure. Because inspectors delve into more of the details of the code through reading it, we expected them to be able to isolate more defects, but they don’t (because they find fewer than the testers). We can also conclude at a reasonable risk level (p=0.08) that testers detect different defects compared to inspectors and with statistical significance show that they isolate different defects (p=0.05). The descriptive statistics indicate that testers and inspectors identify and isolate the same amount of critical defects. Testers, however find more defects overall.
There are several lessons to be learned from this: it may not be useful to compare both techniques in general, because they behave differently. Rather, one might use a combination of both techniques or select one over the other based on a fault model for which one technique behaves more favorably than another. In this regard our results are contrary to Laitenberger’s premise [11] that inspection and testing do not complement each other. In the experiment of Basili and Selby [1] the results were different for different categories of subjects, hence they do not give any overall conclusion. So et al [20] report testing and inspection as complementary, although inspection is considered more cost-effective, which is much in line with this experiment.

Since the picture given by various experiments is so non-homogeneous, further work must include attempts to understand the mechanisms behind the observed differences. By investigating in more depth how the techniques interact with defects of different kinds, more knowledge can be gained on when to use the different techniques.

Acknowledgement

We thank the students who participated in the experiment. We are also grateful to the Fulbright commission for making Dr. Runeson’s sabbatical visit to Washington State University possible. Thanks to Dr. Thomas Thelin and Carina Andersson for reviewing earlier versions of this paper.

References

Appendix: Post-test questionnaire results