Defect Detection Efficiency: Test Case Based vs. Exploratory Testing

Juha Itkonen, Mika V. Mäntylä and Casper Lassenius
Helsinki University of Technology, Software Business and Engineering Institute
P.O. BOX 9210, FIN-02015 TKK, Finland
firstname.lastname@tkk.fi

Abstract

This paper presents a controlled experiment comparing the defect detection efficiency of exploratory testing (ET) and test case based testing (TCT). While traditional testing literature emphasizes test cases, ET stresses the individual tester's skills during test execution and does not rely upon predesigned test cases. In the experiment, 79 advanced software engineering students performed manual functional testing on an open-source application with actual and seeded defects. Each student participated in two 90-minute controlled sessions, using ET in one and TCT in the other. We found no significant differences in defect detection efficiency between TCT and ET. The distributions of detected defects did not differ significantly regarding technical type, detection difficulty, or severity. However, TCT produced significantly more false defect reports than ET. Surprisingly, our results show no benefit of using predesigned test cases in terms of defect detection efficiency, emphasizing the need for further studies of manual testing.

1. Introduction

Many different techniques, tools and automation strategies have been developed to make testing more efficient. Despite the wide variety of proposed solutions, the fundamental challenge of software testing—revealing new defects in freshly developed software or after major modifications—is in practice still largely dependent on the performance of human testers doing manual testing.

While test automation is becoming increasingly popular due to, e.g., approaches like Test-Driven Development and eXtreme Programming [1, 8, 9], empirical research shows that companies typically perform very little automated testing [3] and most new defects are found by manual testing. The role of automation is emphasized in regression testing and it is best viewed as a way of removing the enactment of simple and repetitive tasks from human testers in order to free up time for creative manual testing [3, 11, 14]. Interestingly, manual testing and especially test execution practices have been fairly little studied in the software engineering community. Testing research has focused on techniques for test case design, selection and prioritization, as well as on optimizing automated testing. However, we do not know, i.e., what factors affect the efficiency of manual testing, and how, or what practices industrial testers find useful. Previous research shows that aspects such as testers' skills and the type of the software have as strong an effect on test execution results as the test case design techniques [18].

We think that test execution is not a simple mechanic task of executing completely specified test cases, which can be easily carried out by a novice employee, outsourced, or even completely automated. Instead, testers' skills and knowledge are likely to be important also during test execution. Indeed, often testers use test cases primarily as a means of structuring and guiding their work.

Recently, practitioner literature has discussed the idea of testing without using predesigned test cases, so called *exploratory testing* (ET) [16]. Reports on exploratory testing have proposed that ET, in some situations, could be even orders of magnitude more efficient than test case based testing [5]. Other claimed benefits of ET include the ability to better utilize testers' creativity, experience and skills, lower documentation overhead and lower reliance on comprehensive documentation [5, 6, 16, 20, 22, 26].

Considering the claims stated in the practitioner literature, we decided to carry out an experiment to test a simple question: *Do testers performing manual functional testing with predesigned test cases find more or different defects compared to testers working without predesigned test cases?*

The rest of this paper is structured as follows. Section 2 reviews existing research on test case based and exploratory testing. Section 3 presents the experimental design and data analysis methods. Section 4 presents the experimental results, which are discussed in Section 5, along with a discussion of the limitations of this study. Finally, in Section 6 we present the conclusions and outline future research directions.

2. Background

Testing in the software engineering literature is considered a process based upon the design, generation, selection, and optimization of a set of *test cases* for testing a certain system or functionality. Various methods and techniques have then been developed that help determine what test cases to execute [10, 13, 23]. The underlying assumption is that given the right set of documented test cases prior to testing, testing goals can be achieved by more or less mechanically executing the test cases. However, this view is problematic for at least three reasons. First, empirical studies of testing techniques show that there are many other factors than the technique used to design or select test cases that explain the effectiveness and efficiency of testing. These include, e.g., properties of the actual software being tested, the types of the actual defects in the tested software and the experience, skills, and motivation of testers [18]. Second, the actual importance of documenting test cases *before* executing the tests is unknown. Third, practitioner reports of testing approaches not based on a predesigned set of test cases claim results that are clearly comparable to those obtained using more formal techniques for test case design [5, 22, 26].

2.1. Experiments on testing techniques

Several experimental studies have been conducted in order to compare test techniques to each other, essentially looking at how to most efficiently build and execute an "optimal" set of test cases. These studies are reviewed in [18]. Juristo concludes that existing knowledge is limited, somewhat conflicting and lack a formal foundation [18].

Kamsties and Lott found that time taken to find a defect was dependent on the subject [19]. Basili and Selby, instead, found that the fault rate depended on the software under study, and that the defect detection rate was unrelated to tester experience [7]. Wood et al. found defect detection rate to depend on the type of faults in the program [27]. These studies show that factors other than the test case design technique can have significant effects on the testing results.

One conclusion that can be drawn from the existing studies is that more faults are detected by combining individual testers than techniques [18]. This is important because it shows that the results of test execution vary significantly despite the test case design strategy used. Wood et al. found that combined pairs and triplets of individual testers using the same technique found more defects than individuals [27]. The testers seem to find different defects even though using the same technique. Similar results were reported also for code reading and structural testing techniques.

Possible reasons for the variation in the results are many. Individual testers might execute the documented tests differently; the testers' ability to recognize failures might be different; or individual testers might end up with different test cases even though using the same test case design technique.

However, designing the test cases beforehand and writing them down in a test case specification document is only one way of applying defect detection strategies. A strategy can be applied with or without detailed test cases and it is hard to understand the effects of the detailed documentation and the effects of the applied strategy.

2.2. Industrial experiences

While only a few studies have looked at industrial practice, they show that test cases are seldom rigorously used and documented in industrial settings. Instead, practitioners report that they find test cases difficult to design and often quite useless [2, 3, 16].

In practice, it seems that test case selection and design is much left to the individual testers: "The use of structured approaches to V&V is sparse. Instead, the selection of test cases is very much based on the experience of the staff." [3]. Even more interesting is the finding that "On the other hand, no one reported particular problems that can be traced back to the lack of structured methods specifically" [3].

It seems, that large amount of testing in industry is performed without applying actual testing techniques or, e.g., any formal test adequacy criteria. Reasons for this can be many, but it shows the importance of studying and improving also these less formal testing approaches.

2.3. Exploratory testing

Exploratory testing is an approach that does not rely on the documentation of test cases prior to test execution. This approach has been acknowledged in software testing books since the 1970's [23]. However, authors have usually not presented actual techniques or methods for performing exploratory testing; instead treating it as an 'ad hoc' or error guessing method. Furthermore, exploratory testing lacks scientific research [16]. While test case design techniques set the theoretical principles for testing, it is too straightforward to ignore all the factors that can affect testing activities during test execution work.

In the context of verifying executable specifications Houdek et al. [15] have performed a student experiment comparing reviews, systematic testing techniques and the exploratory (ad-hoc) testing approach. The results showed that the exploratory approach required less effort, and there was no difference between the techniques with respect to defect detection effectiveness. None of the studied techniques alone revealed a majority of the defects and only 44% of the defects were such that the same defect was found by more than one technique.

Some research on exploratory testing can be found in end-user programming context. Rothemel et al. [25] reported benefits of supporting exploratory testing tasks by a tool that is based on formal test adequacy criteria. Phalgune et al. have found that oracle mistakes are common and should be taken into account in tools supporting end-user programmer testing [24]. Oracle mistakes, meaning that a tester judges incorrect behaviour correct or vice versa, could be an important factor affecting the effectiveness of exploratory testing and should be studied also in the professional software development context.

Even though the efficiency and applicability of exploratory testing lacks reliable research, there are anecdotal reports listing many benefits of this type of testing. The claimed benefits, summarized in [16] include, e.g., effectiveness, the ability to utilize tester's creativity and non-reliance on documentation [5, 6, 20, 22, 26]. Considering the claimed benefits of exploratory testing and its popularity in industry, the approach seems to deserve more research. The exploratory approach lets the tester freely explore without being restricted by pre-designed test cases. The aspects that are proposed to make exploratory testing so effective are the experience, creativity, and personal skills of the tester. These aspects affect the results, and some amount of exploratory searching and learning exists, in all manual testing; perhaps excluding the most rigorous and controlled laboratory settings. Since the effects of exploratory approach and the strength of those effects have not been studied and are not known, it is hard to draw strong conclusions on the performance of manual testing techniques.

We recognize that planning and designing test cases can provide many other benefits besides defect detection effectiveness. These include, e.g., benefits for test planning, test coverage, repeatability, and tracking. In this paper, however, we focus only on the viewpoint of defect detection effectiveness.

3. Methodology

In this section, we describe the research problem and the experimental design.

3.1. Research problem and questions

We study the effects of using predesigned test cases in manual functional testing at the system level. Due to the scarce existing knowledge, we focus on one research problem: *What is the effect of using predesigned and documented test cases in manual functional testing with respect to defect detection performance?*

Based on existing knowledge we can pose two alternative hypotheses. First, because almost all research is focused on test case design issues we could hypothesise that the results are better when using predesigned test cases. Second, from the practitioner reports and case studies on exploratory testing we could draw a hypothesis that results are better when testing without predesigned test cases. The research questions and the hypotheses of this study are presented below.

Research question 1: How does using predesigned test cases affect the number of detected defects?

Hypothesis $H1_0$: There is no difference in the number of detected defects between testing with and without predesigned test cases.

Hypothesis $H1_1$: More defects are detected with predesigned test cases than without predesigned test cases.

Hypothesis $H1_2$: More defects are detected without predesigned test cases than with predesigned test cases.

Research question 2: How does using predesigned test cases affect the type of defects found?

Hypothesis $H2_0$: There is no difference in the type of the detected defects between testing with and without predesigned test cases.

Research question 3: How does using predesigned test cases affect the number of false defect reports?

Hypothesis $H3_0$: There is no difference in the number of produced false defect reports between testing with and without predesigned test cases.

False defect reports refer to reported defects that cannot be understood, are duplicates, or report non-existing defects. This metric is used to analyze how using test cases affects the quality of test results.

3.2. Experimental design

We used a one-factor block design with a single blocking variable [17]. We also used the empirical research guidelines presented by Kitchenham et al. [21], as applicable to our context.

The study was performed as a student experiment on the software testing and quality assurance course at Helsinki University of Technology in November 2005.

Participation in the experiment was a compulsory part of the course. The subjects were randomly divided into two groups, both of which performed similar test sessions with and without test cases. The experiment consisted of three separate phases: preparation, session 1, and session 2. In the preparation phase, each subject designed and documented test cases for the feature set that was allocated for test case based testing for the group. All subjects, regardless of which testing approach they first utilized, designed and submitted their test cases according to the same schedule. The subjects designed the test cases without supervision and got to use as much effort as they required for the preparation phase. Note that each student designed the test cases only for the test case based testing, they did not prepare test cases for the other feature set that was tested using exploratory approach. An overview of the experimental arrangements is shown in Table 1.

Table 1. Experiment arrangements

Phase	Group 1	Group 2
Preparation	Test cases for feature set A	Test cases for feature set B
Testing session 1	Test case based testing Feature set A	Exploratory testing Feature set A
Testing session 2	Exploratory testing Feature set B	Test case based testing Feature set B

The subjects were instructed to use the techniques they had learned in the course, i.e. equivalence class partitioning, boundary value analysis and combination testing. The source documentation for the test case design was the User's Guide for the tested software. The subjects' task was to cover all functionality that was documented in the User's Guide concerning their allocated feature set.

The subjects' performance in the experiment affected their course grade: the quality of the test cases and their performance in test execution were evaluated by the course assistants. The grading was based on the subjectively evaluated quality of their predesigned test cases and defect reports and the number of defects they found during the controlled test sessions.

Testing session 1 took place one week after the submission deadline for the test case designs, and testing session 2 one week after the first session. All subjects participated in both sessions, but the ordering of the test approaches was different for the two groups. The structure and length of both controlled testing sessions were exactly the same, as shown in Table 2. The subjects of Group 1 and Group 2 performed the sessions at the same time in different computer classrooms. In both sessions, the same application, the open source text editor JEdit, was tested, but the tested feature set was different in session 1 and session 2. Note that in this design the two testing approaches were compared, not the results of the two groups or the two sessions.

Table 2. Testing session phases

Phase	Length	Description
Session setup	15 min	Introduction and guidelines Downloading and starting the correct variant of the jEdit application.
Functional testing	90 min	Focused testing following the exploratory or test case based approach. Writing the test log and reporting all found defects.
Survey and submitting the reports and logs	Around 10 min	Short survey form is filled in and defect reports and test logs collected.

3.2.1. Experimental units. The experimental units were two variants of version 4.2 of the jEdit text editor. Both variants were created from the same application release by artificially seeding defects into the application at the source code level and then recompiling.

We had three major reasons for selecting jEdit. First, we wanted the tested software to be as realistic as possible, not an unrealistically small and simple application. Second, it had to be possible to seed defects into the application. Third, the application domain had to be familiar to the students without special training.

JEdit, while being a fairly simple text editor, has a far too wide and complicated functionality to be tested as a whole, even superficially, in the 90 minute scope of the test sessions of this experiment. Therefore, we chose two distinct and restricted feature sets for testing; Feature set A for Session 1 and Feature set B for Session 2. We created two different variants of the tested software in which we artificially seeded defects: In variant A we seeded 25 defects in Feature set A, and in variant B we seeded 24 defects in Feature set B. Naturally, the number of seeded defects was not the total number of defects in the software as any real software is usually far from defect free. This was also the case with JEdit. The variants with seeded defects were not available to the subjects before the test sessions. The normal open source version of the software was of course available to the subjects beforehand, and they could familiarize themselves with the features and utilize the software when designing their test cases.

3.2.2. Factors and blocking variables. The factor in this experiment is the applied testing approach. The factor has two alternatives: test case based testing (TCT) and exploratory testing (ET).

Blocking variables represent the undesired variations in the experimental design that cannot be eliminated or made constant. In this experiment the only significant blocking variable was the tested feature set, including the actual and seeded defects that could not be kept the same for all elementary experiments. The reason for this is the fact that we wanted to run the experiment twice with each subject—once with both of the factor alternatives—in order to reduce the possible effects of sampling error and increase the sample size. This design meant that there must be two separate testing sessions for each subject. After the first testing session, the subjects are naturally much more familiar with the tested functionality and the behaviour of the application. From the experimental point of view, also the defects in the tested variant of the software must be considered public after the first testing session. This forced us to use different feature sets and different sets of seeded defects in the two testing sessions. In addition, the actual defects that exist in the tested software variant affect the test results: the total number and types of defects differs in the feature sets as does the difficulty of detecting them.

3.2.3. Response variables. This study looked at the defect detection efficiency measured by the number of defects found during a fixed length testing session. Additionally, more insight into the efficiency is gained by considering the proportions of different defect types and severities as well as the number of false defect reports produced during a testing session.

3.2.4. Subjects. The final number of subjects who performed both phases of the experiment and thus were included in the experimental data, was 79. The subjects were randomly assigned into two groups; Group 1 (39 students) and Group 2 (40 students). We collected demographic data on the subjects to characterize them in terms of experience in software development and testing, phase of M.Sc. studies, etc. 27 subjects had no previous experience in software engineering and 63 had no previous experience in testing. 8 subjects had one year and 4 subjects had two years testing experience. Only four subjects reported having some sort of training in software testing prior to taking the course. The demographic data is summarized in Table 3. The credits in the Table 3 refer to Finnish study credits. The M.Sc. degree requires 160 credits.

Table 3. Characteristics of the subjects

Characteristic	$\bar{x}$	$\tilde{x}$	σ
Study year	4,8	4,0	1,8
Credits	107,9	110,0	41,6
Sw dev experience (years)	2,0	1,0	2,7
Testing experience (years)	0,5	0,0	1,1

$\bar{x}$ = mean, $\tilde{x}$ = median, and σ = standard deviation

3.2.5. Parameters. The most important parameters in this experiment are the individual properties of the student subjects, the type of the software under test, the time available for test execution, the tools used, the testing environment, and the training given.

The major undesired variation originates from the individual properties of the student subjects, e.g., experience in software engineering, amount of studies, prior training in software testing, and individual skills. These variations are handled by two means. First, all subjects performed the experiment two times, once using each of the testing approaches. Second, the subjects were randomly assigned into two groups that applied the two approaches in opposite orders. The two groups were used for the sole purpose of randomizing the application order of the two approaches, and the testing assignments in this experiment were individual tasks for each of the subjects.

The tested software was the same throughout the experiment. The available time used for test execution was fixed to 90 minutes. The testing tools and environment was an identical PC workstation with a Windows XP environment in university computer classrooms for each elementary experiment.

3.2.6. Internal replication. In this experiment, the elementary experiment corresponds to one subject applying one of the two factor alternatives to test one of the two variants of the tested software. We had a paired design where 79 subjects all replicated the elementary experiment two times, once using each of the two testing approaches (factor alternatives). This adds up to a total of 158 internal replications and 79 paired replications with both alternatives and a single subject.

3.2.7. Training and instructions. The subjects were trained to use the test case design techniques before the experiment. The training was given in lecture format and the training material consisted of lecture slides, chapters in the course text book [12], and excerpts from another software test design book [13]. The training was supported by multiple choice questionnaires.

Instructions for the assignments were given on the course web site. In the testing sessions, the subjects got printed instructions on the session arrangements, but no instructions on testing techniques or strategies. Testers using test cases got only brief instructions to follow their predesigned test cases. Exploratory testers got a brief charter that listed the tested functionality and instructed them to focus testing from an average user's viewpoint and additionally to pay attention to issues that may be problematic for an advanced user.

3.3. Data collection and analysis

We collected data in three ways. First, subjects submitted the predesigned test cases in electronic format. Second, in the testing sessions the subjects filled in test logs and defect report forms. Third, after each session the subjects filled in a survey questionnaire.

The number of defects detected by ET and TCT groups were compared using the t-test. In addition, we used multi-factorial analysis of variance (ANOVA) to control for and understand the effect of the different feature sets, and the possible interactions between the feature set and the testing approach. Interaction would mean that the effect of a testing approach is not similar in the case of the two different feature sets. The t-test and ANOVA are both parametric methods and thus assume that the analyzed data is normally distributed and at least on interval scale. We can assume that the defect count data is roughly normally distributed and it is measured on a ratio scale.

To analyze the differences in the defect types we represent the defect distributions of the ET and TCT groups and perform significance analysis using the Mann-Whitney test that is a non-parametric alternative to the t-test. Finally, to analyze the number of false reports we used the Mann-Whitney test to analyze the significance of the difference between the two approaches. The t-test could not be used for analyzing defect distributions or number of false reports as the data did not have a normal distribution.

The data analysis was performed using the SPSS software package.

4. Results

In this section we present the collected data and the results of the experiment based on the statistical analysis of the data.

4.1. Defect counts

The main response variable in this experiment was the number of defects a subject detected during a 90-minute fixed-length testing session. The defect count data is summarized in Table 4.

Table 4. Summary of defect count data

Testing approach	Feature set	Number of defects	Found defects per subject	
			$\bar{x}$	σ
ET	A	44	6,275	2,172
	B	41	7,821	2,522
	Total	85	7,038	2,462
TCT	A	43	5,359	2,288
	B	39	7,350	2,225
	Total	82	6,367	2,456

$\bar{x}$ = mean and σ = standard deviation

The number of defects refers to how many different individual defects all subjects together found. Since the feature sets were different, the number of individual defects found in each is different. The total numbers of individual detected defects in feature sets A and B were 53 and 48, respectively. Figure 1 contains the box-plots of the data, where the boxes contain 50% of the data points. There are two extreme values in the data of Feature set B. The absolute mean defect counts for the ET and TCT approaches were 7,038 and 6,367 respectively, the difference showing 0,671 defects more in the ET approach, which using the two-tailed t-test is not statistically significant (0,088). For feature sets A and B, the differences between the ET and TCT defect counts were 0,916 and 0,471 respectively. There was no difference in the number of detected *seeded* defects between the approaches. The ET approach detected more *real* (non-seeded) defects.

In this experiment, the effects of learning between the two testing session rounds cannot be separated from the effects of the two feature sets because feature set A was used solely in the first session and feature set B in the second one. This means that we cannot say if the higher reported defect counts in the second testing session are caused by learning or by the type of the features and defects under test. In the subsequent discussion when we talk about the effect of feature sets we mean the combined effect of the feature set and subjects' learning.

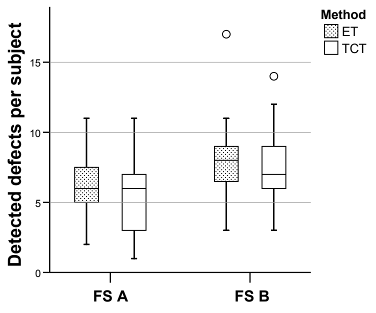

Figure 1. Defect counts

4.2. Effects of testing approach and feature set

We used two different feature sets in this experiment. Although we tried to select similar feature sets to have comparable results, it is clear that the differences in the feature sets could have an effect on the number of defects found. The mean defect count from feature set A was 5,817 and from feature set B 7,585. If we had used completely custom-made "laboratory" software, it would have been possible to better control for the number of defects. However, as we used real world software, we face the problem of having two feature sets where unequal numbers of defects were detected, and where the total number of defects is unknown. Thus, we needed to control for the interaction effect of the feature set.

Table 5. Effect of approach and feature set

Source	F	Sig.
Testing approach	3,57	0,061
Feature set	23,25	0,000
Testing approach * Feature set	0,37	0,544

We used multi-factorial ANOVA to control for the effect of the feature set and to get a better picture of how the feature set in combination with the testing approach factor affects the results. This leads to a 2x2 factorial design, two factors with two levels (alternatives) each. The summary of the results of the ANOVA analysis is presented in Table 5, in which we can see the significance values for both the feature set and the defect detection technique. The effect of the feature sets is statistically significant with a value of 0,000. The effect of the testing approach has a significance value of 0,061. Thus, we can see that the effect of the testing approach is stronger when the feature set effect is controlled for, but it is still not statistically significant.

Based on the ANOVA analysis it is possible to analyze possible interactions between the two factors. In Table 5 we can see that the interaction effect of the testing technique and the feature set has a significance value of 0,544. This means that in this case there was no considerable interaction effect present. In Figure 2 the mean defect counts are plotted for the four combinations of the two factors. This analysis indicates that we have an effect for both testing approach and feature set, but no interaction between the factors.

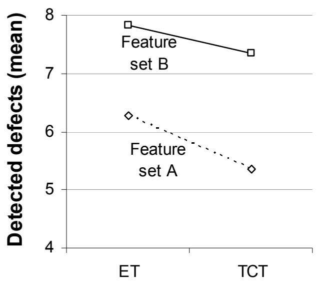

Figure 2. Defect count interaction effect

4.3. Detection difficulty, types, and severities

The distributions of defect type and severity can be used to understand the differences between the two testing approaches. The primary author classified all defect reports according to three dimensions: type, severity, and detection difficulty. Type indicates the technical details of each defect, e.g., usability, performance, documentation. Severity means the defect's impact on the end user. The distribution of the defects according to this classification is presented in Tables 6-8.

Table 6 characterizes the defects based on the detection difficulty. A mode 0 defect means that the defect is immediately obvious to the tester, e.g., a missing button. A mode 1 defect (single-mode defect) requires one action of the tester in order to cause a failure and reveal the defect, e.g., save a file to find out that some part of the file is not saved. The double-mode and triple-mode defects require a combination of 2 and 3 actions or inputs in order to cause failure and get the defect detected. With respect to the difficulty of detection, there is no clear difference between the approaches.

In Table 6 we can see that ET found more defects in all classes of detection difficulty. The most notable differences were for mode 0 and mode 3 defects, for which ET found 29% and 33% more defects than TCT. However, the Mann-Whitney U test shows the differences to be statistically insignificant for all classes.

Table 6. Detection difficulty distribution

Mode	ET	TCT	ET/TCT	Total
0 = easiest	120	93	129 %	213
1	327	320	102 %	647
2	89	75	119 %	164
3 = hardest	20	15	133 %	35
Total	556	503	111 %	1059

Table 7 shows the defects categorized based on their technical type. From the table we can see that there are no radical differences in the number of defects with different technical types. ET found 10% more wrong function defects, 43% more GUI defects, and 280% more usability problems than TCT.

Table 7. Technical type distribution

Type	ET	TCT	ET/TCT	Total
Documentation	8	4	200 %	12
GUI	70	49	143 %	119
Inconsistency	5	3	167 %	8
Missing function	98	96	102 %	194
Performance	39	41	95 %	80
Technical defect	54	66	82 %	120
Usability	19	5	380 %	24
Wrong function	263	239	110 %	502
Total	556	503	111 %	1059

However, for the usability defects, we must note that the absolute numbers are very small. On the other hand, TCT found 22% more technical defects. The

Mann-Whitney U test shows that the only significant difference is for the Usability defects (p=0,006).

Table 8 shows the defects categorized based on their severities. From the table we can see that ET found 64% more negligible defects, 32% more minor defects, and 14% more normal defects. TCT found 5% more severe and 2% more critical defects. The only significant difference according to the Mann-Whitney U test was for minor defects (p=0,038).

Table 8. Severity distribution

Severity	ET	TCT	ET/TCT	Total
Negligible	23	14	164 %	37
Minor	98	74	132 %	172
Normal	231	203	114 %	434
Severe	153	160	96 %	313
Critical	51	52	98 %	103
Total	556	503	111 %	1059

We must emphasise that by using repeated Mann-Whitney tests we are likely to come up with statistically significant values by chance. Thus, the reader should be cautious with inferences based on the statistically significant values for the defect type and severity classes presented in this section.

4.4. False defect reports

The data of false defect reports, meaning defect reports that are incomprehensible, duplicate or reported a non-existent defect, are summarized in Table 9. TCT produced on average 1,05 more false reports than ET.

Due to a non-normal distribution, we used the Mann-Whitney U test that showed that the effect of testing approach is highly significant with a two-tailed significance of 0,000.

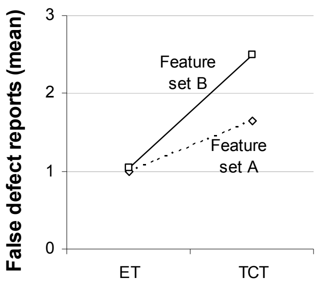

Figure 3. False defect interaction effect

Figure 3 illustrates the interaction between the effects of the testing approach and the feature set with respect to false defect report count. From the figure we can see the main effect between ET and TCT, ET having less false reports. There also is an interaction effect as more defect reports are reported by TCT testers with feature set B than with feature set A.

Table 9. False defect counts

Testing approach	Feature set	False defects per subject	
		$\bar{x}$	σ
ET	A	1,00	1,396
	B	1,05	1,191
	Total	1,03	1,291
TCT	A	1,64	1,564
	B	2,50	1,867
	Total	2,08	1,767

$\bar{x}$ = mean and σ = standard deviation

5. Discussion

This section summarizes the results and reflects the findings in the light of existing research and knowledge. Additionally, we outline the limitations of this research as well as discuss future research.

5.1. Answering the research questions

5.1.1. Research question 1. *How does using predesigned test cases affect the number of detected defects?* In this experiment, the subjects found less defects when using predesigned test cases. Statistical test showed that there is an 8,8% probability that this result is obtained by chance. Thus, the difference between the two approaches was not statistically significant, and does not allow rejecting the null hypothesis that assumes there is no difference in the number of detected defects when testing with or without test cases.

Although we cannot reject the null hypothesis, the results strengthen the hypotheses of the possible benefits of exploratory testing. Based on the results of this study, we can conclude that an exploratory approach could be efficient, especially considering the average 7 hours of effort the subjects used for test case design activities. This means that testing with predesigned test cases in this study took on average 8,5 hours, whereas testing without test cases took on average 1,5 hours. Still, the defect detection rates of the two approaches were not different. The benefits of exploratory testing have been proposed to be based on the experience and skills of the testers. In this experiment, the subjects had received some training regarding test case design techniques, but did not have any specific techniques or methods for exploratory testing. Thus, at least in the context of this experiment, the exploratory approach is more efficient as no time is spent on creating test cases.

5.1.2. Research question 2. *How does using predesigned test cases affect the type of found defects?* We analyzed the differences in the types of the detected defects from three viewpoints; severity, type, and detection difficulty. Based on the data, we can conclude that testers seem to find more of both the most obvious defects, as well as the ones most difficult to detect when testing without test cases. In the terms of defect type, the testers found more user interface defects and usability problems without test cases. More technical defects were found using test cases. When considering defect severity, the data shows that more low severity defects were found without test cases. The statistical significance of the differences in all these defect characterizations is low. We must be cautious of drawing strong conclusions based on the defect classification data even though the results show a significant difference in the numbers of usability and minor defects detected between the two approaches.

The differences in the defect types and severities suggest that testing without test cases tend to produce larger amounts of defects that are obvious to detect and related to user interface and usability issues. These differences could be explained by the fact that test cases are typically not written to test obvious features and writing good test cases for testing many details of a graphical user interface is very laborious and challenging. On the other hand, subjects testing without test cases found more defects that were difficult to detect, which supports the claims that exploratory testing makes better use of tester's creativity and skills during test execution. The higher amount of low severity defects detected without test cases suggests that predesigned test cases guide the tester to pay attention on more focused areas and thus lead to ignoring some of the minor issues.

5.1.3. Research question 3. *How does using predesigned test cases affect the number of false defect reports?* The purpose of this research question was to provide an understanding on the effects of the two approaches from the test reporting quality viewpoint. The data in section 4.4. shows that testers reported around twice as many: 2,08 vs. 1,03, false defect reports when testing with test cases than when testing without test cases. This difference is statistically significant.

This issue raises the more general question of the consequences of following predesigned test cases in manual test execution. Test cases are used to guide the work of the tester and more studies are needed to better understand how different ways of documenting tests and guiding testers' work affect their behaviour in performing the tests and the results of testing efforts.

5.2. Limitations

The main threats to external validity of this study are using students as subjects, the time-boxed testing sessions, and variations in the applied testing techniques. It is not obvious how the results of a student experiment can be generalized to the industrial context, but we have presented the data on the professional and academic experience of our subjects in Section 3. The subjects' lack of testing experience might have affected the quality of the test cases as well as the performance in exploratory testing tasks.

In this experiment we had strictly time-boxed and controlled testing sessions, which is good for internal validity, but raises some questions about how typical this kind of setting would be in industry. Such strict restriction as the 90-minute time-box places might not be typical in industry, but short calendar time for testing in general is very typical restriction. Testing approaches that can adapt to testing time restrictions will be highly relevant for the industry.

The subjects of the experiment were instructed to use the trained black-box testing techniques for the test case design, but we could not control that the subjects actually used those techniques properly. For the exploratory testing sessions we cannot determine if the subjects used the same testing principles that they used for designing the documented test cases or if they explored the functionality in pure ad-hoc manner. For this reason it is safer to assume the ad-hoc manner to hold true.

The threats to internal validity of this study include the combined learning effect and the effect of the tested feature set. We could not analyze how good test case designers our subjects were and how much the quality of the test cases affected the results and how much the actual test execution approach. In addition, it seems that all subjects could not execute all the test cases they had designed during the time-boxed session.

6. Conclusions and future work

This paper makes four contributions. First, we identify a lack of research on manual test execution from other than the test case design point of view. It is obvious that focusing only on test case design techniques does not cover many important aspects that affect manual testing. Second, our data showed no benefit in terms of defect detection efficiency of using predesigned test cases in comparison to an exploratory testing approach. Third, there appears to be no big differences in the detected defect types, severities, and in detection difficulty. Fourth, our data indicates that test case based testing produces more false defect reports.

Studying factors that affect defect detection effectiveness and efficiency is an important direction for future research. At least most of the reported test case design techniques are based on theories for effectively revealing defects in software, but these have been studied only using predesigned and documented test cases. More research is required to study the effect of predesigned test cases in comparison to other approaches to manual testing.

Planning and designing test cases can provide many other benefits besides defect detection efficiency, e.g. benefits in test planning, traceability, test coverage, repeatability and regression testing, tracking and controlling the progress of testing efforts, and test reporting. Using an exploratory approach to testing instead of predocumented test cases requires some other approach for planning, structuring, guiding and tracking the testing efforts, e.g., session-based test management [6, 22]. Approaches for managing exploratory testing are a natural candidate for further research on this area.

In the inspection and review literature, a lot of research focuses on review execution. Ways of performing inspection meetings and approaches to document reading have been widely studied [4]. Similar approaches for manual testing have not been presented. However, both reviewing and manual testing are human activities with the intent of revealing defects and quality issues in the target artifact or software system. These issues should be studied in the area of manual testing.

References

[1] Abrahamsson, P., J. Warsta, M. T. Siponen, and J. Ronkainen, "New Directions on Agile Methods: A Comparative Analysis", in *Proceedings of ICSE*, 2003, pp. 244-254.

[2] Ahonen, J. J., T. Junttila, and M. Sakkinen, "Impacts of the Organizational Model on Testing: Three Industrial Cases", *Empirical Software Engineering*, vol. 9(4), 2004, pp. 275-296.

[3] Andersson, C. and P. Runeson, "Verification and Validation in Industry - A Qualitative Survey on the State of Practice", in Proceedings of ISESE, 2002, pp. 37-47.

[4] Aurum, A., H. Petersson, and C. Wohlin, "State-of-the-art: software inspections after 25 years", *STVR*, vol. 12(3), 2002, pp. 133-154.

[5] Bach,J., "Exploratory Testing", in *The Testing Practitioner*, Second ed., E. van Veenendaal Ed., Den Bosch: UTN Publishers, 2004, pp. 253-265.

[6] Bach, J., "Session-Based Test Management", *STQE*, vol. 2, no. 6, 2000,

[7] Basili, V. R. and R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies", *IEEE TSE*, vol. 13(12), 1987, pp. 1278-1296.

[8] Beck, K., *Test Driven Development by Example*, Addison-Wesley, Boston, 2003.

[9] Beck, K., "Embracing Change With Extreme Programming", *Computer*, vol. 32(10), 1999, pp. 70-77.

[10] Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1990.

[11] Berner, S., R. Weber, and R. K. Keller, "Observations and Lessons Learned form Automated Testing", in *Proceedings of ICSE*, 2005, pp. 571-579.

[12] Burnstein, I., Practical Software Testing, Springer-Verlag, New York, 2003.

[13] Copeland, L., *A Practitioner's Guide to Software Test Design*, Artech House Publishers, Boston, 2004.

[14] Fewster, M. and D. Graham, *Software Test Automation*, Addison-Wesley, Harlow, England, 1999.

[15] Houdek, F., T. Schwinn, and D. Ernst, "Defect Detection for Executable Specifications - An Experiment", *IJSEKE*, vol. 12(6), 2002, pp. 637-655.

[16] Itkonen, J. and K. Rautiainen, "Exploratory Testing: A Multiple Case Study", in *Proceedings of ISESE*, 2005, pp. 84-93.

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

[18] Juristo, N., A. M. Moreno, and S. Vegas, "Reviewing 25 years of Testing Technique Experiments", *Empirical Software Engineering*, vol. 9(1-2), 2004, pp. 7-44.

[19] Kamsties, E. and C. Lott, "An Empirical Evaluation of Three Defect-Detection Techniques", in *Proceedings of the 5th ESEC*, 1995,

[20] Kaner, C., J. Bach and B. Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York, 2002.

[21] Kitchenham, B. A., et al., "Preliminary guidelines for empirical research in software engineering", *IEEE TSE*, vol. 28(8), 2002, pp. 721-734.

[22] Lyndsay, J. and N. van Eeden, "Adventures in Session-Based Testing", 2003, Accessed 2007 06/26, http://www.workroom-productions.com/papers/AiSBTv1.2.pdf

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

[24] Phalgune, A., C. Kissinger, M. M. Burnett, C. R. Cook, L. Beckwith, and J. R. Ruthruff, "Garbage In, Garbage Out? An Empirical Look at Oracle Mistakes by End-User Programmers", in *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005, pp. 45-52.

[25] Rothermel, K., C. R. Cook, M. M. Burnett, J. Schonfeld, Green, Thomas R. G., and G. Rothermel, "WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation", in *Proceedings of ICSE*, 2000, pp. 230-239.

[26] Våga,J. and S. Amland, "Managing High-Speed Web Testing", in *Software Quality and Software Testing in Internet Times*, D. Meyerhoff, B. Laibarra, van der Pouw Kraan,Rob and A. Wallet Eds., Berlin: Springer-Verlag, 2002, pp. 23-30.

[27] Wood, M., et al., "Comparing and Combining Software Defect Detection Techniques: A Replicated Empirical Study", *ACM SIGSOFT Software Engineering Notes*, vol. 22(6), 1997, pp. 262-277.