T. Thelin, P. Runeson, C. Wohlin, T. Olsson and C. Andersson, "How Much Information is Needed for Usage-Based Reading? - A Series of Experiments", Proceedings 1st International Symposium on Empirical Software Engineering, pp. 127-138, Nara, Japan, October 2002. Best paper award, and selected and extended for a special issue of Empirical Software Engineering: An International Journal.

How much Information is Needed for Usage-Based Reading? – A Series of Experiments

Thomas Thelin[1], Per Runeson[1], Claes Wohlin[2], Thomas Olsson[1] and Carina Andersson[1]

[1]*Dept. of Communication Systems*
Lund University
Box 118, SE-22100 Lund, Sweden
{thomast, perr, thomaso, carinaa}@telecom.lth.se

[2]*Dept. of Software Eng. and Computer Science*
Blekinge Institute of Technology
Box 520, SE-372 25 Ronneby, Sweden
claes.wohlin@bth.se

Abstract

Software inspections are regarded as an important technique to detect faults throughout the software development process. The individual preparation phase of software inspections has enlarged its focus from only comprehension to also include fault searching. Hence, reading techniques to support the reviewers on fault detection are needed. Usage-based reading (UBR) is a reading technique, which focuses on the important parts of a software document by using prioritized use cases. This paper presents a series of three UBR experiments on design specifications, with focus on the third. The first experiment evaluates the prioritization of UBR and the second compares UBR against checklist-based reading. The third experiment investigates the amount of information needed in the use cases and whether a more active approach helps the reviewers to detect more faults. The third study was conducted at two different places with a total of 82 subjects. The general result from the experiments is that UBR works as intended and is efficient as well as effective in guiding reviewers during the preparation phase of software inspections. Furthermore, the results indicate that use cases developed in advance are preferable compared to developing them as part of the preparation phase of the inspection.

1. Introduction

Software inspections have emerged over the last 25 years as a key technique to detect and hence remove faults throughout the software development process [1]. Researchers have over the years proposed several different ways of improving software inspections. The improvements include new inspection processes, for example, represented by n-fold inspection [12] and phased inspections [10]. It also includes changes to the different steps in the inspection processes, for example, new reading techniques [2] and whether an inspection meeting is needed or not [22]. Finally, other improvements include support to the inspection process, for example, fault content estimations [15]. The objectives of this paper are in general to contribute to the improvement of reading techniques and to study a new reading technique. More specifically, the main aim is to contribute to the evaluation of a reading technique called usage-based reading.

The user perspective in software development is acknowledged and valued in different methods including for example use cases in object-orientation [9] and operational profile testing [13]. However, the user perspective can also be introduced into software inspections, where the individual preparation may be conducted with a user-oriented approach. Such a method has been proposed and evaluated in a series of experiments. The method is denoted usage-based reading. It was initially presented by Olofsson and Wennberg [14], and has since been extended and evaluated in two main studies [19][20].

This paper contributes with a third study as well as a presentation of the series of experiments. The latter includes the lessons learned through a planned series of experiments, where the studies have built upon each other to create a body of knowledge with respect to usage-based reading. The first experiment focused upon comparing use case driven inspections with prioritized use cases versus randomly ordered use cases. After having found out that the prioritization, which constitutes a key part in usage-based reading, was significantly better, usage-based reading was compared with checklist-based reading. Usage-based reading was found to be significantly better than using a checklist. The third experiment looks at whether the reviewers perform better in the preparation phase if they develop use cases as part of the inspection or if it is better to utilize pre-developed use cases in the inspection. The third experiment also studies usage-based reading as part of the inspection process, i.e. a meeting is also held in the third study. It is concluded that usage-based reading is both effective and efficient.

The paper is outlined as follows. Usage-based reading is described in Section 2, and an overview of the series of experiments is presented in Section 3. The third experiment is then discussed in detail in Section 3.3 to Section 7. The artefacts for the experiment are presented in Section 4. In Sec-

tion 5, the planning of the experiment can be found. The operation of the experiment is discussed in Section 6 and the analysis is presented in Section 7. A discussion of the third experiment and the series of experiments can be found in Section 8. Finally, conclusions are presented in Section 9.

2. Usage-Based Reading

The individual preparation of software inspections has enlarged its focus from only comprehension (initially proposed by Fagan) [6] to also comprise fault searching. Hence, support to the reviewers on how to detect faults is needed. Therefore, different reading techniques have been developed, for example, defect-based reading [16] and perspective-based reading [2]. Usage-based reading (UBR) is one such reading technique, which focus on the quality of the product from a user's point of view.

The cornerstones of UBR are *use cases* and *prioritization*. Use cases are utilized to guide reviewers through a software document during inspection. The use cases are prioritized (for example by using the Analytic Hierarchy Process (AHP) [17]) in an order of importance from users' requirements on the system developed. Hence, reviewers using UBR focus on the important parts first, leading to the important faults are found. The most important faults are denoted *critical* in the paper and refer to the faults that a user of a system thinks are most important.

The main purpose of UBR is to focus inspections on users' needs, much in the same way as operational profiling in testing [13]. The reviewers applying UBR follow the prioritized used cases and check the software artefact under inspection. During inspection, the reviewers need to go through the artefact actively, although they do not need to actively develop the use cases. However, in this paper it is investigated how much information that is needed in order to apply UBR, see Section 3.3.

There are some other reading techniques that utilize use cases during fault searching. Among these are traceability-based reading (TBR) [21] and the user perspective in perspective-based reading (PBR) [3]. TBR is a reading technique aimed for inspecting object-oriented design specifications. The user perspective in PBR is based on active development of use cases during inspections. Hence, the use cases are developed on the fly and the reviewers are supported with a scenario of how the development should be carried out. A benefit of using already developed use cases is that they may be prioritized by a user. In addition, Dunsmore et al. compare a use case approach with checklist-based reading (CBR) and structured reading [5]. The use cases are based on sequence diagrams for object-oriented design and are not prioritized. The results indicate that CBR is the best reading technique of the three for object-oriented code inspections.

3. Usage-Based Reading Experiments

In order to evaluate the UBR technique, a series of three experiments was planned and conducted. The subjects performed inspections of design documents using different reading techniques. The experiments were conducted in academical settings with students at software engineering programmes in their third or fourth years of studies.

The sequence of experiments is summarized in Table 1. The first and second experiments are presented in Section 3.1 and Section 3.2 respectively, and the third experiment is presented in Section 3.3 and onwards.

Table 1: The main research questions in the series of UBR experiments.

	1st	2nd	3rd
Question	Are the reviewers affected by the reading technique?	Is UBR more effective and efficient than CBR?	Is pre-developed use cases needed for UBR?
Answer	Yes, prioritized use cases are more efficient than randomly ordered use cases.	Yes, UBR finds more critical faults than CBR.	Yes, in most cases, reviewers using detailed use cases find more faults.

3.1. First Experiment

The first experiment, which was aimed at investigating the basic principle of UBR, was launched during the fall semester of 2000. 27 students of their third year of the software engineering Bachelor's programme at Lund University (Campus Helsingborg) inspected a high-level design for a taxi management system. The design document contained 37 faults, of which 13 where critical (class A), 13 were important (class B) and 11 were not important (class C) (see Section 4.2 for fault classification). Half of the subjects were given use cases that were prioritized with respect to the impact on the intended users of the system. The other half of the subjects were given the same use cases, but in a random order. The experiment is presented in more detail in [19].

Three main research questions were investigated in the experiment:

- RQ1 – Is UBR effective in finding the most critical faults?

- RQ2 – Is UBR efficient in terms of total number of critical found faults per hour?
- RQ3 – Are different faults detected using different priority orders of use cases?

In Table 6, the efficiency and effectiveness values are presented for the group that used prioritized use cases. Hypotheses were set up and tested. It was concluded that the reviewers applying prioritized use cases were more *efficient* in detecting faults than the reviewers using randomized use cases ($p=0.044$). This was also true for class A faults ($p<0.001$) and for class A&B faults ($p=0.005$). Furthermore, the reviewers applying prioritized use cases were more *effective* in detecting faults than the reviewers using randomized use cases for class A faults ($p=0.002$) and class A&B faults ($p=0.005$). They were *not* significantly more *effective* for all faults. Finally, the reviewers applying prioritized use cases detected different faults compared to the reviewers applying randomized use cases ($p<0.001$).

Hence, we can conclude that reviewers actually perform differently when given a different inspection method. By prioritizing the use cases, reviewers are forced to focus on issues that are of highest importance for the system user, and hence improve the efficiency and effectiveness of the inspection process.

3.2. Second Experiment

The purpose of the second experiment was to compare UBR to checklist-based reading (CBR). CBR is some kind of industry practice, and was used as a baseline to which UBR was compared. The experiment was launched during the spring semester of 2001. 23 students of their fourth year of the software engineering Master's programme at Blekinge Institute of Technology inspected the same high-level design, although the design documented had been updated due to further development of the system (some faults were removed and some injected). The design document contained now 38 faults, of which 13 were critical (class A), 14 were important (class B) and 11 were not important (class C). In the second experiment, half of the subjects were taught the UBR method and were given prioritized use cases. The other half of the subjects were taught the CBR method and were given a checklist, where the items were sorted in a significance order. The experiment is presented in more detail in [20].

Three main research questions were investigated in the experiment, similar to the ones on the first experiment:

- RQ1 – Is UBR more effective than CBR in finding the most critical faults?
- RQ2 – Is UBR more efficient than CBR in terms of total number of critical faults found per hour?
- RQ3 – Are different faults detected when using UBR and CBR?

Hypotheses, similar to the ones in the first experiment were set up and tested. In this experiment, it was concluded that the reviewers applying UBR were more *efficient* in detecting faults than the reviewers using CBR ($p=0.042$, 37% more faults per hour). This was also true for class A faults ($p=0.013$, 100% more class A faults per hour) and class A&B faults ($p=0.016$, 64% more class A&B faults per hour). Furthermore, the reviewers applying UBR were more *effective* in detecting faults than the reviewers using CBR for class A faults ($p=0.036$, 75% more class A faults) and class A&B faults ($p=0.031$, 51% more class A&B faults). They were *not* significantly more *effective* for all faults. Finally, the reviewers applying UBR detected different faults compared to the reviewers applying CBR ($p=0.001$).

From this analysis, we draw the conclusion that the UBR method is more effective and efficient compared to the industry practice method, CBR.

3.3. Third Experiment

The third experiment, which was launched during the fall semester of 2001, investigates how much information is needed in the use cases when performing UBR inspections. It takes time to develop the use cases in detail, and more information in the use case document may cause inconsistencies. Hence, they should not be more detailed than motivated by the use of the use cases.

The rationale of the experiment is to investigate whether less provided information to the reviewers still makes UBR efficient and effective. The question has been raised whether reviewers detect more faults if they develop something actively during inspection (as in PBR and TBR), i.e. whether they are more effective. It is also evaluated whether there are any differences in efficiency and whether they find other and more critical faults from a user's point of view. UBR, as investigated in the previous experiments, utilizes use cases developed before the inspection. Here, we investigate whether it is enough with only the purpose of the use cases and the prioritization. Hence, the comparison is against UBR with purpose and tasks pre-developed (see Section 4.1), and the main research questions are:

- RQ1 – How much information is needed in the use cases for UBR?
- RQ2 – Is UBR with less information as efficient and effective?
- RQ3 – Are different faults detected when actively developing the use cases?

In the following sections, the third experiment is described in detail.

4. Experiment Artefacts

The development of the software documents and the design of the series of experiments have involved seven persons in total. The persons have taken different roles in the development of the experiment package since it was important to develop and design some parts of the experiment independently in order to minimize the threats to the validity of the experiments. The artefacts are briefly described in the following sections. A detailed description of the artefacts and the development of those is provided in [19].

4.1. Documents

The software artefacts in the experiment package are a textual requirements document, use case documents, a design document, and questionnaires. In addition, code and test documents have been developed for the system, but are not used in the experiments. The code was written in the specification and description language (SDL) [7]. The following documents were used in the third experiment.

- Textual Requirements Document – The textual requirements document was written in natural language (English). The document was used as a reference document to know what the system should do.
- Use Case Documents – The use case documents contains 24 use cases. There are two versions, one for the group with pre-developed use cases (utilizing method) and one for the group with only the purpose specified (developing method). The pre-developed use cases were written in task notation [11]. An example of a use case for the taxi management system in task notation is shown to the left in Figure 1; the corresponding use case with less information is shown to the right.

  The prioritization of the use cases was made in terms of the importance from a user's point of view. Three persons acted users and prioritized the use cases before the experiment according to the AHP method [17]. The order of use cases was the same for both methods (utilizing and developing). The fault classification is based on the same criterion for prioritization, i.e. the users are in focus.
- Design Document – The design document (9 pages, 2300 words) consists of an overview of the software modules and communication signals that are sent to and from the modules. In addition, the communication between the system and the users is specified. Furthermore, the design document contains two message sequence charts (MSC) [8], which show signalling between the modules for two different cases.

  The faults injected during development of the taxi system were re-inserted into the design document. The design document was updated between the first and second experiment. In the first experiment, the document contained 37 faults, and in the second and third, it contained 38 faults. Eight faults were seeded by the developer the system. The 30 others were faults made during development of the design document and later found in inspections or testing.
- Experience Questionnaire – A questionnaire with seven questions was used to explore the students' experiences in programming, inspections, SDL, use cases and the application domain. This information was used in the design of the experiments in order to control the different experiences among the students.
- Inspection Questionnaire – After the inspection, a questionnaire was filled in to explore how well the reviewers have followed the specified inspection process and what they thought about the method used.

4.2. Fault Classification

The faults are classified from the point of view of a user. They are divided into three classes depending on the importance for a user, which is a combination of the probability of the fault to manifest and the severity of the fault from a user's point of view.

- Class A faults – The functions affected by these faults are crucial for a user, i.e. the functions affected are important for a user and are often used. An example of this kind of faults is: the operator cannot *log in* to the system.
- Class B faults – The functions affected by these faults are important for a user, i.e. the functions affected are either important and rarely used or not as important but often used. An example of this kind of fault is: the operator cannot *log out* of the system.
- Class C faults – The system will work although these faults are present. An example of this kind of fault is: a signal is confused with another signal in an MSC diagram.

The design document contains 13 class A faults, 14 class B faults and 11 class C faults.

5. Experiment Planning

5.1. Subjects

The experiment was conducted at two universities in Sweden during a period of two weeks. First, the experiment was conducted at Campus Helsingborg (Hbg) at Lund University and then in Ronneby (Rb) at Blekinge Institute of Technology. The experiment was a mandatory part of two

Taxi: Alarm Event (Utilizing method)

Purpose: If a driver, for some reason, feels threatened there is an alarm system which notifies the central of the problems.

Tasks:

1. The driver is in some kind of trouble and hits the alarm button.
2. The voice radio channel to the central is opened from the taxi to the central.
3. The taxi sends exact coordinates to the central every 30 seconds.
4. The channel is open until the central resets the alarm. The voice link is terminated.

Taxi: Alarm Event (Developing method)

Purpose: If a driver, for some reason, feels threatened there is an alarm system which notifies the central of the problems.

Figure 1. An example of a use case written in task notation. The use case describes an alarm event. To the left is an example of a use case that was given to the utilizing groups and to the right is a use case that was given to the development groups, see Section 5.2

courses in verification and validation. The courses included lectures and assignments, and both courses were related to verification and validation of software products and evaluation of software processes. Although the courses have the same name, they do not include exactly the same items. The main difference is that the course in Rb is more research oriented and the course in Hbg is more focused on a test project.

The subjects in Hbg were 34 third-year students at the software engineering Bachelor's programme at Lund University. The students were almost finished with their education and have experience in requirements engineering, use case development, software design and in the particular application domain (taxi management systems). They had participated in, for example, a large-scale software project course and a previous course in requirements engineering, where they developed a requirements document for a taxi management system. This means that the students have good domain knowledge.

The subjects in Rb were 48 fourth-year software engineering Master's students at Blekinge Institute of Technology. Many of the students have extensive experience from software development. As part of their Bachelor degree, they have obtained practical training in software development. Among other things, they have participated in a one-semester project including 15 students. The customers for these projects are normally people in industry, and hence the students have participated in projects close to an industrial situation with changing requirements and time pressure. Several of the Master students also work in industry in parallel with their studies. This means that the students are rather experienced and to some extent comparable to fresh software engineers in industry.

Thus, the difference between the student groups (Hbg and Rb) can be referred to their education, domain knowledge, and industrial experience.

5.2. Variables

Three types of variables are defined for the experiment, *independent*, *controlled* and *dependent* variables. The independent variable is the reading technique used and the controlled variable is the experience of the students. The dependent variables are measures collected to evaluate the effect of the methods.

The reading technique used is either *utilizing* (utilizing use cases) use cases or *developing* (developing use cases) use cases. Both these treatments were used in Hbg and Rb. The abbreviations used in this paper are Util (utilizing use case), Dev (developing use cases), Hbg (Helsingborg), Rb (Ronneby) and the combination of these, i.e. Util-Hbg, Dev-Hbg, Util-Rb and Dev-Rb.

5.3. Hypotheses

The hypotheses of the experiment are set up to evaluate the amount of information needed in order to utilize UBR. The hypotheses are expressed in terms of efficiency and effectiveness of finding critical faults from a user's perspective.

The main alternative hypotheses of the experiment are stated below. These are evaluated for all faults, class A faults and class A&B faults.

- H_{Eff} – There is a difference in *efficiency* (i.e. found faults per hour) between the reviewers utilizing (Util) pre-developed use cases and the reviewers who develop (Dev) use cases.
- H_{Rate} – There is a difference in *effectiveness* (i.e. rate of faults found) between the reviewers utilizing pre-developed use cases and the reviewers who develop use cases.

- H_{Fault} – The reviewers utilizing pre-developed use cases *detect different faults* than the reviewers who develop use cases.

5.4. Design

The subjects were divided into two groups, Util and Dev. This was carried out in Hbg and Rb, separately. Using the controlled variable (experience), the students were divided into three groups (at each place) and then randomized within each group, resulting in 17 students in the Util-Hbg group, 17 students in the Dev-Hbg group, 23 students in the Util-Rb group and 25 students in the Dev-Rb group. One student in each group in Hbg and one in Util-Rb were removed from the analysis, since they did not complete all parts of the experiment.

The experiment data are analyzed with descriptive analysis and statistical tests [23]. The collected data were checked for normal distribution. Since no such distribution could be demonstrated using normal probability plots and residual analysis, non-parametric tests are used. The Mann-Whitney test is used to investigate hypotheses H_{Eff} and H_{Rate} and a chi square test is used to test H_{Fault} [18].

The significance value of rejecting the hypotheses is set to 0.05 for all tests. In addition to the chi-square test, a partial correlation analysis is used to measure the degree of similarity of the faults that the reviewers found.

The Kruskal-Wallis test is used to test whether there are statistical differences in the time data. If there is a significant difference, the multiple comparison procedure is used, see Siegel and Castellan [18].

An alternative would be to use a non-parametric two-way ANOVA (Friedman's test). The two independent variables would then be the student groups (Hbg and Rb) and the reading technique used. However, Friedman's test assumes balanced data, i.e. equal number of students in Hbg and Rb, which is not the case. Thus, statistical tests are used within each place and descriptive analysis is used between the students groups, i.e. Hbg and Rb.

5.5. Threats to Validity

A key issue when performing experiments is the validity of the results. Is the study designed and performed in a sound and controlled manner? Are the statistical tests used correctly? Are the issues under investigation really investigated in the study? To which domain are the results possible to generalize? In this section, the threats are analyzed related to four groups of threats: Conclusion validity, internal validity, construct validity and external validity [23].

Conclusion validity concerns the relation between the treatments and the outcome of the experiment. The threats related to the statistical tests used in the experiment are considered being under control as non-parametric tests are used, which do not require a certain underlying distribution. Threats with respect to the subjects are also limited since (1) there are two subject groups which both treatments are assigned to, and (2) the subject groups are rather homogeneous; the subjects have attended the same education programs (at each place).

Internal validity of the experiment concerns the question whether the effect is caused by the independent variables or by other factors. The social threats about imitation of treatment, compensation for treatment etc. are limited since the subjects had nothing to gain from the actual outcome of the experiment. The grading of the courses was only based on their participation in the experiment, not on their performance. The threat of selection is also under control, as the experiment is a mandatory part of a course. There is one threat of ambiguity of the direction of causal influence, concerning the time spent reading and the method used. This is further analyzed in Section 7.1.

Construct validity concerns the ability to generalize from the experiment result to the concept behind the experiment. The experiment is the third in a series, where the same method has been used as one treatment in all the three experiments, giving very much the same results with respect to effectiveness and efficiency (see Section 8). The performance measures as such are standard: faults found per hour and rate of faults found. Furthermore, the inspection questionnaire is used as a cross-check of the results, to reduce the mono-method bias. Hence, we conclude that the treatment applied actually is the difference between the different subject groups.

External validity concerns the ability to generalize the results to industry practice. The largest threat is that students are used as subjects. However, the students are in their third or fourth year of software engineering studies and hence close to start working in industry. The members of one of the subject groups are familiar with the application domain, which is industry-like, as they have developed a requirements specification for a similar, but more extensive system, in a previous course. The comparison between the two subject groups enables blocking with respect to domain knowledge and educational background. The inspected document is the same in this experiment as in the two former, which is a threat to the external validity. On the other hand, it strengthens the internal validity.

In summary, the threats are not considered large in this experiment. As it is a follow-up experiment, some issues can be monitored over all the experiments, and the analysis shows that the results are stable.

6. Experiment Operation

The experiment was run in September 2001. The experiment was conducted during one day both in Hbg and Rb, see schedule in Table 2. However, since the students in Hbg had participated in a course where they developed a requirements document for a taxi management system, they only had a brief introduction. Thus, the general introduction (A in Table 2) was only performed for the students in Rb. The inspection (B in Table 2) was divided into three parts, preparation, inspection and questionnaire times. The aim of these was to read the documents briefly (about 20 minutes), inspect the design document, and answering questions about the reading technique used (about 15 minutes).

Table 2: Schedule for the Experiment. Note that (A) was only performed in Ronneby.

Time		Util group	Dev group
Day 1 (A) (10.15 a.m. - 11.00 a.m.)	45 min.	General introduction to the Taxi Management System	
Day 1 (A) (11.15 a.m. - 12.00 a.m.)	45 min.	Introduction to UBR (Utilizing)	Introduction to UBR (Development)
Day 1 (B) (13.15 p.m. - 17.00 p.m.)	3 h 45 min.	Preparation and Inspection Questionnaire (15 minutes)	

7. Analysis

7.1. Preparation and Inspection Time

The reviewers logged the time for preparation (read through the documents) and inspection. In Table 3, the mean and standard deviation values of preparation, inspection and total times are shown for the four groups. For preparation time, no significant difference can be observed. However, Dev-Rb differs significantly against the others considering inspection time and total time. The standard deviations are all in the same ranges, indicating the dispersions are equal for the different groups. Hence, Dev-Rb, which is one of the groups that developed use cases, used significantly more inspection time. The other group that developed use cases, Dev-Hbg, did not use more time.

Furthermore, the groups used different number of use cases during the inspection. The use case document consists of 24 use cases. The median of the number of use cases used is 24 for all groups except Dev-Rb, which has a median value of 19.

Table 3: Mean and standard deviation values for preparation and inspection time (minutes).

		Hbg		Rb	
		Util	**Dev**	**Util**	**Dev**
Mean	**Preparation**	36.2	32.6	31.7	28.2
	Inspection	100.9	102.5	107.0	139.1
	Total	137.1	135.1	138.7	167.3
Std. Dev.	**Preparation**	11.7	9.1	8.6	9.8
	Inspection	21.0	24.5	26.3	24.8
	Total	25.6	25.9	29.2	26.2

In addition to preparation and inspection times, the reviewers also logged the time when a fault was found. In Figure 2, the cumulative number of faults found is plotted versus the time for an average reviewer (an average reviewer is created by standardizing the faults found by all reviewers using a certain method with the number of reviewers using that method). The plots show that an average Util reviewer was more efficient than an average Dev reviewer. For class A faults, the difference is small and the slope does not increase, indicating that Dev reviewers were at least as efficient as Util reviewers, especially if development time of use cases is taken into consideration.

The reviewers using the Util method show similar patterns, while the reviewers using the Dev method do not show similar patterns. There are the same patterns for class A and A&B faults as for all faults. In these cases, the slopes of Util-Hbg, Util-Rb and Dev-Rb do not increase. This indicates that after the first faults are found (first use cases are developed) there is a constant time difference between these groups. The difference between Dev-Hbg and the others increases, indicating that Dev-Hbg needed more time to develop the use cases.

Consequently, there is a larger difference between the treatments in Hbg than Rb. The treatments that developed use cases needed more time, and this time was not used in the Hbg case. This can be seen by looking at the inspection times and also in the plots, where the curve for Dev-Rb does not decrease in the end.

7.2. Efficiency and Effectiveness

In this section, the efficiency and effectiveness of the different treatments are evaluated. The efficiency and effectiveness are evaluated for both places (Hbg and Rb) together and separately as described in Section 5.4.

In Figure 3, the efficiency is shown for all faults, class A and class A&B faults. The two first box plots of each class of faults show Util and Dev when the reviewers from Hbg

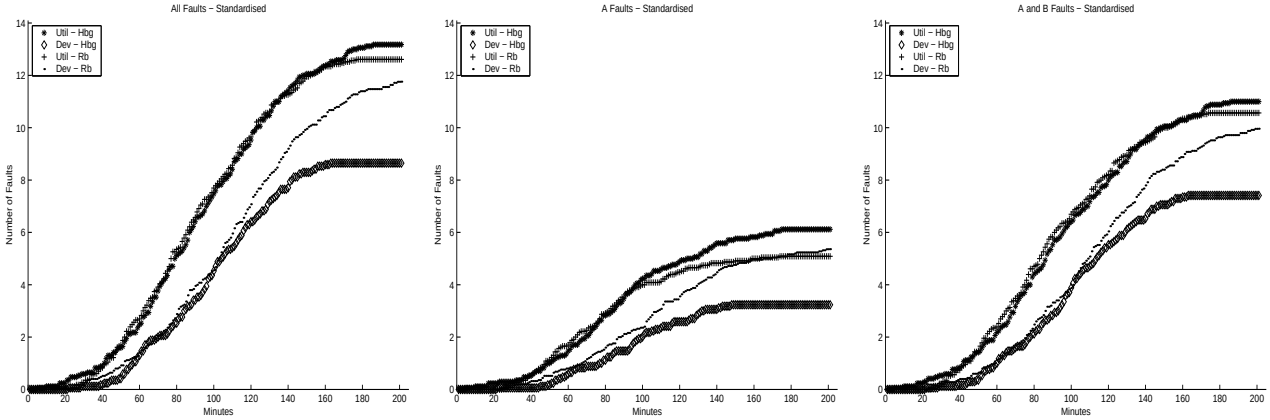

Figure 2. The figures show the faults found over time for an average reviewer. The plots are, from left to right, all faults, class A faults and class A&B faults. Preparation and inspection time are included in the figures.

and Rb are combined; the next two are reviewers from Hbg and the last two are reviewers from Rb. The same order is present in Figure 4, where the effectiveness values are presented. In total, the Util groups were more efficient and effective for all faults, class A and class A&B faults. As discussed in the previous section, the box plots show that there is larger difference between groups in Hbg than in Rb.

In general, the order of efficiency as well as effectiveness is, from high to low, Util-Hbg, Util-Rb, Dev-Rb, Dev-Hbg. Thus, more faults are found when pre-developed use cases are utilized in inspections. Note that for class A faults, Dev-Rb is more effective than Util-Rb.

The hypothesis testing is performed as described in Section 5.4. The p-values of the significance tests for H_{Eff} and H_{Rate} are presented in Table 4. These show that the efficiency and effectiveness are significantly higher for Util than for Dev in Hbg, but not in Rb.

Table 4: P-values for the hypotheses

	Efficiency		Effectiveness	
	Hbg	**Rb**	**Hbg**	**Rb**
All	0.006 (S)	0.043 (S)	0.002 (S)	0.462 (–)
A	<0.001 (S)	0.550 (–)	0.001 (S)	0.416 (–)
A+B	0.008 (S)	0.097 (–)	0.006 (S)	0.582 (–)

Regarding *efficiency* in Hbg, significant differences occur for all faults, class A and class A&B faults. In Rb, there is only a significant difference between Util and Dev for all faults. Hence, there is no significant difference obtained between the treatments in Rb for class A and Class A&B faults.

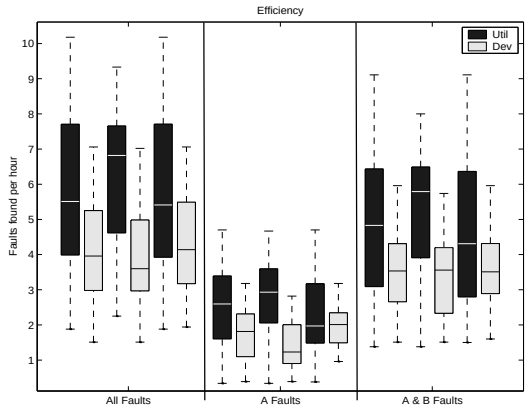

Figure 3. Efficiency for all faults, class A and class A&B faults. The box plot shows, from left to right, both groups, Hbg and Rb.

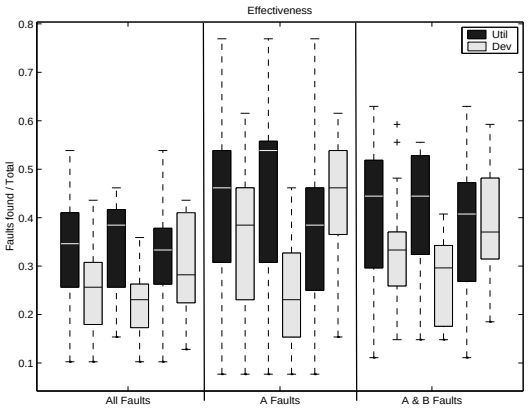

Figure 4. Effectiveness for all faults, class A and class A&B faults. The box plot shows, from left to right, both groups, Hbg and Rb.

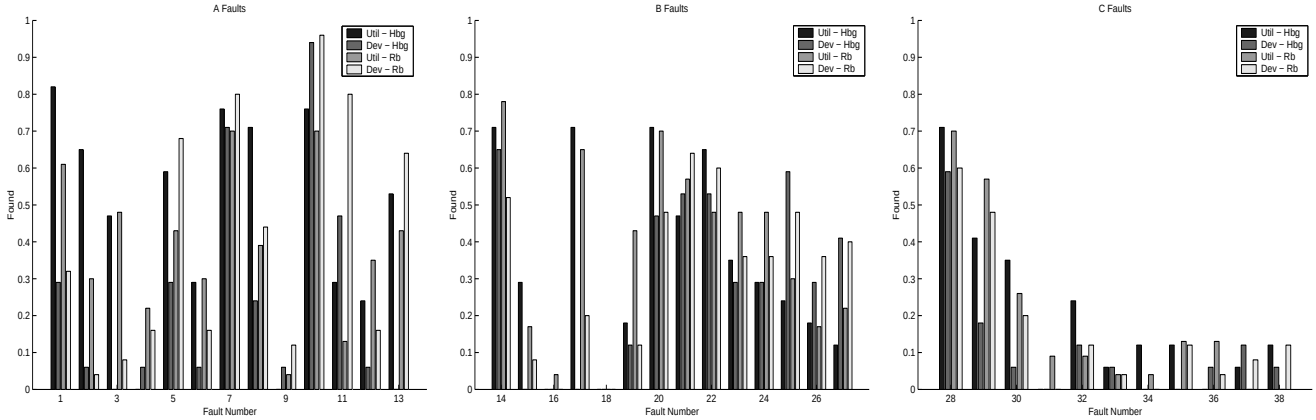

Figure 5. Bar plots of the percentage of faults found by the groups. From left to right, class A faults, class B faults and class C faults.

Regarding *effectiveness* in Hbg, the Mann-Whitney test shows that there are significant differences for all faults, class A and class A&B faults. There is, however, no significant difference between the treatments in Rb.

Consequently, there is a large difference between Util-Hbg and Dev-Hbg and only a small between Util-Rb and Dev-Rb. This may depend on two factors; one, more time is used per use case by the reviewers in the Dev-Rb group, and two, there may be a difference between the students' capability in creating use cases between Hbg and Rb.

7.3. Faults

The faults were categorized in three classes depending on the level importance from a user's point of view. UBR is designed to detect the most important faults, since the use cases were prioritized by using the same criterion as the faults. Hence, the probability of finding class A faults should be higher than class B and in its turn class C faults. Bar plots of distribution the faults that each treatment found in the experiment are shown in Figure 5.

The plots show that there is a higher probability to find class A and class B faults than class C faults. The plots also indicate that different faults are found depending on whether the reviewers utilize use cases (Util) or develop use cases (Dev). In order to evaluate whether the reviewers detected different faults (H_{Fault}), a chi-square test is used. The test is performed for the different places separately, i.e. Util-Hbg against Dev-Hbg and Util-Rb against Dev-Rb. The result of the chi-square tests shows that they find different faults ($p<0.001$ for Hbg and $p=0.006$ for Rb).

To measure the degree of similarities among the faults found, a partial correlation measure is used, see Table 5. The partial correlations verify the result, which shows that there is only small correlation between the faults that the different treatments found. However, there is a larger correlation between the faults that the Util groups found as well as the two Dev groups, at the different places (Hbg and Rb). This indicates that the faults they found, depend more on the treatments used and less on the students.

Table 5: Partial Correlations between the faults found by the groups.

	Util-Hbg	**Util-Rb**	**Dev-Hbg**	**Dev-Rb**
Util-Hbg	1	**0.767**	-0.148	0.256
Util-Rb		1	0.225	-0.016
Dev-Hbg			1	**0.751**
Dev-Rb				1

7.4. Questionnaire Data

After the reviewers had finished the individual inspection, they filled in a questionnaire. The aim of the questionnaire is to explore how well the reviewers followed the inspection process and what they think about the method.

Figure 6 shows the distribution of how difficult the reviewers found the inspection method. There is no large difference between the Util and Dev groups. However, it seems like the reviewers in Rb found the inspection method more difficult to use. The reason for this may be because the reviewers in Hbg have better domain knowledge.

Figure 7 shows how the reviewers rank their conformance with the inspection process. All of the participants seem to have followed the process fairly well. The trend of the data seems to be that the reviewers in Hbg generally felt they followed the process closer than Rb, and that the Util groups followed the process closer than the Dev groups. Note that the inspection process includes everything carried out during inspection, for example, log clock time, estimate

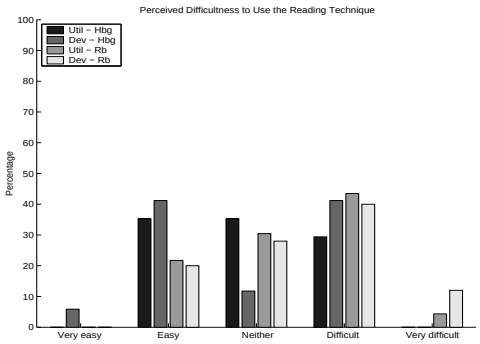

Figure 6. Bar plot of the perceived difficulty of using the reading technique.

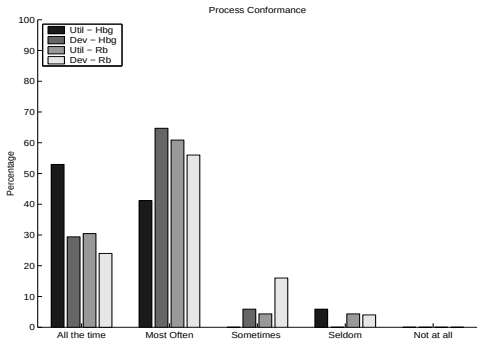

Figure 7. Bar plot of the conformance of the inspection process.

the risk of the faults found and utilizing or developing use cases.

Furthermore, the perceived difficulty and conformance with the process seem to coincide with treatment and group although the difference is small. Since the reviewers in Hbg have more domain knowledge, a prior expectation was that they would find it easier and thereby follow the process better. It was also expected that the Dev groups would find it more difficult since they had to develop the tasks during inspection.

A question of whether they would like to use the method again was asked. In the Hbg group, 88% would like to use the method again, and in the Rb group, 67% would. No differences could be observed between the Dev and Util groups in the different places.

Consequently, the subjective data reveal no significant differences between the treatments or the students. It also seems like they have followed the process fairly well and understood the reading technique they used.

8. Discussion

The results of the hypotheses are summarized below.

- H_{Eff} – For both subject groups combined (Hbg and Rb), the Util method was more efficient. In Hbg, the Util method was significantly more efficient in Hbg for all faults, class A and A&B faults. In Rb, the Util group was significantly more efficient for all faults.
- H_{Rate} – For both subject groups combined, the Util method was more effective. In Hbg, the Util method was significantly more effective in Hbg for all faults, class A and A&B faults. In Rb, the Util group was not significantly more effective in Rb.
- H_{Fault} – The reviewers using the Util method found different faults than the reviewers using the Dev method. The faults that the reviewers in the Util groups (Util-Hbg vs. Util-Rb) found are correlated and the faults that the reviewers in the Dev (Dev-Hbg vs. Dev-Rb) group found are correlated.

The rationale for investigating these hypotheses is whether UBR can be used without developing the use cases prior to the inspection session and whether other faults are found when a more active reading method is used. Utilizing pre-developed use cases (Util) leads to that the reviewers become more focused on detecting faults. On the other hand, developing use cases during inspection (Dev) could lead to that other faults are found, since they have to comprehend the document in greater detail to develop the use cases. Travassos et al. [21] argue that it requires semantic processing when reviewers need to actively develop artefacts during inspection. Thus, the semantic processing could be one reason of detecting more faults. In this experiment, the reviewers did not extract use cases from the design. They were given the purpose of every use case and had to develop the tasks, which also requires semantic processing, probably more than only following already developed use cases.

If the development time of the use cases is considered, it may be questioned whether it is more efficient to develop them beforehand. The development of the use cases was estimated to about 30 minutes per use case for the taxi system. Hence, if software organizations are not using use cases in their development, they should not be developed for UBR purposes only. Note that other parameters should also be taken under consideration when deciding whether the use cases should be developed beforehand or not. For example, how difficult the system is to understand, the importance of the use cases and the cost-effectiveness of software inspections.

The different treatments (Util and Dev) detected different faults. Furthermore, there is a correlation between the faults that were found by the reviewer groups using the same method. This means that there is a higher probability

to find the same faults with a specific method. Especially the results of the Util groups are similar.

A consequence of the discussion above is that UBR may be more efficient if a hybrid approach is designed. An extension of UBR is to develop detailed use cases of the ones that are considered crucial, less detailed of the important ones and no development of the least important use cases. Hence, the use cases could be categorized in groups depending on priority. Then, the use cases could be developed with different details considering priority.

The data reveal larger differences between the methods in Hbg than in Rb. An explanation of this may be that the Dev-Rb reviewers used more inspection time than the other groups (about 30 minutes more). The reviewers have been differently introduced to the methods at the places depending on their experiences and courses taken before the experiment. The differences between the places, according to their own answers in the experience questionnaire, are mostly on the language used (SDL) and taxi management systems. About programming, inspections and use cases, the subjects think they have similar knowledge. Although this is the case, the Dev groups did not perform equally well. A reasonable explanation may be that the students in Hbg are less experienced. The students in Hbg were third-year Bachelor students and in Rb they were fourth-year Master students. Furthermore, the education programme is different and some of the students in Rb have industrial experience. Consequently, less experience may result in less efficiency and effectiveness, especially for the Dev groups, since they had to develop use cases.

The impact of experience is discussed by Basili et al. [3], when analyzing conducted defect-based reading experiments [16]. They argue that experienced reviewers are needed in order to utilize the benefits of DBR. This since the reviewers have to develop artefacts during inspection. The impact of experience on inspections is also investigated and discussed by Cheng and Jeffery [4]. They argue that experienced reviewers do not need much pre-developed information when inspecting. It is more important to have a decomposition strategy, which experienced reviewers are able to set on their own. A similar effect is shown in this experiment when comparing the Dev groups at different places. The reviewers in Rb are more experienced than reviewers in Hbg. Hence, they may have an advantage when use cases need to be developed during inspection.

In all UBR experiments, the UBR variant with pre-developed use cases has been used (called Util in this paper). The design document has been the same for all experiments, except for some updates between the first and second experiment. Hence, a comparison of the mean efficiency and effectiveness values are possible in order to compare the students' performance of the three experiments. For all faults, both the efficiency and effectiveness values are equal or slightly higher for the experiment presented in this paper than the previous two, see Table 6. This indicates that the students have performed at least as well as in the other experiments.

Table 6: Efficiency and effectiveness values of UBR for the three experiments.

		Exp 1	Exp 2	Exp 3 Hbg	Exp 3 Rb
Efficiency (faults/hour)	All faults	5.3	5.6	6.0	5.6
	class A	2.6	2.7	2.8	2.3
	class B	1.8	1.8	2.2	2.5
	class C	0.9	0.9	1.0	0.9
Effectiveness	All faults	0.29	0.31	0.34	0.32
	class A	0.43	0.43	0.47	0.39
	class B	0.30	0.31	0.35	0.39
	class C	0.17	0.20	0.20	0.19

9. Conclusion

Three experiments of UBR are described in this paper with a focus on the analysis of the third. The common result is that UBR works as intended and is efficient as well as effective in guiding reviewers during the preparation phase of software inspections. The prioritization of UBR forces reviewers to focus on the important parts of the software artefact during inspection, which in its turn makes the inspection more efficient and effective in detecting important faults.

From the first experiment, it is concluded that the reviewers find different faults using prioritized use cases compared to randomly ordered use cases. Furthermore, UBR (prioritized use cases) is significantly more effective and efficient than randomly ordered use cases in finding faults of high importance for a user.

From the second experiment, it is concluded that UBR is significantly better than CBR in terms of both effectiveness and efficiency in finding the faults that affect the user the most. The results show that reviewers applying UBR are more efficient and effective in detecting the most critical faults from a user's point of view than reviewers using CBR.

From the third experiment, it is concluded that is more efficient to use pre-developed use cases for UBR. However, there is a trade-off of whether the use cases should be developed beforehand or on the fly during inspection. The benefit of the latter is that other faults are found, since the reviewers are not that controlled as in the case where they utilize already developed use cases. Hence, there is no clear answer

of how much information that is needed for UBR. It depends on the experience of the reviewers, the software organization and effort used for inspections.

The series of experiments presented in this paper shows that UBR has the potential to become an important reading technique. However, more research is needed and there are several areas that should be considered in order to further develop and investigate UBR. Among these are replications, time-controlled reading (add time limits to the use cases), reading techniques for inspection meetings and case studies in software organizations.

Acknowledgement

This work was partly funded by The Swedish National Agency for Innovation Systems (VINNOVA), under a grant for the Center for Applied Software Research at Lund University (LUCAS).

References

[1] Aurum, A., Petersson, H. and Wohlin, C., "State-of-the-Art: Software Inspections after 25 Years", to appear in *Software Testing, Verification and Reliability*, 2002.

[2] Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. and Zelkowitz, M. V., "The Empirical Investigation of Perspective-Based Reading", *Empirical Software Engineering: An International Journal*, 1(2):133-164, 1996.

[3] Basili, V. R., Shull, F. and Lanubile, F., "Building Knowledge through Families of Experiments", *IEEE Transactions on Software Engineering*, 25(4):456-473, 1999.

[4] Cheng, B. and Jeffery, R., "Comparing Inspection Strategies for Software Requirement Specifications", *Proc. of the 8th Australian Software Engineering Conference*, pp. 203-211, 1996.

[5] Dunsmore, A., Roper, M. and Wood, M., "Further Investigations into the Development and Evaluation of Reading Techniques for Object-Oriented Code Inspection", *Proc. of the 24th International Conference on Software Engineering*, pp. 47-57, 2002.

[6] Fagan, M. E. "Design and Code Inspections to Reduce Errors in Program Development", *IBM System Journal*, 15(3):182-211, 1976.

[7] ITU-T Z.100 *Specification and Description Language*, SDL, ITU-T Recommendation Z.100, 1993.

[8] ITU-T Z.120 *Message Sequence Charts*, MSC, ITU-T Recommendation Z.120, 1996.

[9] Jacobson, I., Christerson, M., Jonsson, P. and Övergaard G. *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, USA, 1992.

[10] Knight, J. C. and Myers, A. E., "An Improved Inspection Technique", *Communications of ACM*, 36(11):50-69, 1993.

[11] Lauesen, S., *Software Requirements – Styles and Techniques*, Addison-Wesley, UK, 2002.

[12] Martin, J. and Tsai, W. T., "N-Fold Inspection: A Requirements Analysis Technique", *Communications of ACM*, 33(2):225-232, 1990.

[13] Musa, J. D., *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, McGraw-Hill, USA, 1998.

[14] Olofsson, M. and Wennberg, M., "Statistical Usage Inspection", Master's Thesis, Dept. of Communication Systems, Lund University, CODEN: LUTEDX (TETS-5244)/1-81/(1996)&local 9, 1996.

[15] Petersson, H., Thelin, T., Runeson, P. and Wohlin, C., "Capture-Recapture in Software Inspections after 10 Years Research – Theory, Evaluation and Application", CODEN:LUTEDX (TETS-7184) / 1-16 / 2002 & local 14, Dept. of Communication Systems, Lund University, 2002. http://www.telecom.lth.se/Personal/thomast/reports/Crc10Years_TechnicalReport.pdf. (submitted to *Journal of Systems and Software*, 2001).

[16] Porter, A., Votta, L. and Basili, V. R., "Comparing Detection Methods for Software Requirements Inspection: A Replicated Experiment", *IEEE Transactions on Software Engineering*, 21(6):563-575, 1995.

[17] Saaty, T. L., *The Analytic Hierarchy Process*, McGraw-Hill, USA, 1980.

[18] Siegel, S. and Castellan, N. J., *Nonparametric Statistics for the Behavioral Sciences*, McGraw-Hill, Singapore, 1988.

[19] Thelin, T., Runeson, P. and Regnell, B., "Usage-Based Reading – An Experiment to Guide Reviewers with Use Cases", *Information and Software Technology*, 43(15):925-938, 2001.

[20] Thelin, T., Runeson, P. and Wohlin, C., "An Experimental Comparison of Usage-Based and Checklist-Based Reading", submitted to *IEEE Transactions on Software Engineering*, 2002, (shorter version available at http://www.cas.mcmaster.ca/wise/wise01/ThelinRunesonWohlin.pdf).

[21] Travassos, G., Shull, F., Fredericks, M., Basili, V. R., "Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality", *Proc. of the International Conference on Object-Oriented Programming Systems, Languages &* Applications, 1999.

[22] Votta, L. G., "Does Every Inspection Need a Meeting?", *Proc. of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering, ACM Software Engineering Notes*, 18(5):107-114, 1993.

[23] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B. and Wesslén, A., *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publisher, USA, 2000.