Experimentation with Usage-Based Reading

Thomas Thelin[1], Magnus Erlansson[1], Martin Höst[1], and Claes Wohlin[2]

[1]Dept. of Communication Systems, Lund University
Box 118, SE-221 00 Lund, Sweden
thomas.thelin@telecom.lth.se, martin.host@telecom.lth.se

[1]Dept. of Software Engineering 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 from a user's point of view in a software document by using prioritized use cases. UBR has been evaluated in two previously conducted experiments, which investigate the prioritization of UBR and compare UBR against checklist-based reading (CBR). This chapter presents two controlled experiments with UBR on requirements and design specifications. The experiments include individual preparation and inspection meeting, i.e. the first steps of the traditional inspection process. For the requirements inspection, UBR is evaluated against CBR, and for the design inspection, the amount of information needed in the use cases are studied. The studies were conducted in different environments with a total of about 100 students. The result from these experiments shows that UBR is not better than CBR for requirements inspections. Results from the experiment on design inspection 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 are regarded as an important technique to detect faults throughout the software development process [1]. 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.

Several reading techniques have been proposed and empirically evaluated for software inspections [2]. The first structured reading technique was checklist-based reading (CBR), suggested by Fagan [1][3]. CBR provides the individual reviewers with a checklist of the faults that should be looked for. Examples of other reading techniques are defect-based reading [4] and perspective-based reading [5], which both have the goal to minimize the overlap among the faults found and thus increase the effectiveness. In addition, they also force the reviewers to actively design software artifacts during inspections.

1.1 Usage Based Reading

Usage-based reading (UBR) stems from an idea from Wohlin and Ohlsson, which is published in [6]. The purpose was to let the expected usage govern the inspection. The motivation behind the method was that faults that affect the user of the software the most are crucial to find, and hence an inspection method setting the user in focus was needed. Usage-based reading (UBR) is a reading technique, which focuses on the important parts of a software document by using prioritized use cases [7].

Software inspections are carried out in order to find faults at an early stage of the software development process. The general steps of an inspection session are [8]:

1) Presentation of material: The reviewers are presented with the material they should inspect.
2) Individual inspections: All reviewers individually read the material in order to find faults.
3) Meeting: A meeting is held where the reviewers meet and summarize their findings. The result of this step is a compiled list of faults.
4) Update: The inspected material is updated by the author. In some cases a new inspection round is held.

UBR supports mainly step 2 of the process. In UBR, the individual work is guided by a set of prioritized use cases. When the reviewer inspects a document, he/she goes through and manually executes a set of use cases and at the same time checks the document. The use cases are prioritized according to their importance for a user of the system, and the idea is that important faults, from users' point of view, should be found during inspection. UBR is described in more detail in [7][9].

In this chapter, two experiments on UBR are presented. The experiments evaluate UBR for requirements inspections and design inspections, respectively. The experiments were carried out as two controlled experiments in the ESERNET framework. In both experiments, the subjects were 3rd and 4th year Bachelors and Masters students in courses in software engineering at university level. The results of the experiments could be input to further controlled experiments and industrial experiments.

1.2 Previous work on UBR

UBR has been evaluated in a series of experiment, where the main purpose is to investigate whether UBR is an efficient and effective reading technique to be used for requirements and design inspections. Before the experiments presented in this chapter, two experiments on UBR have been conducted. In summary, the experiments show positive results in favor for UBR:

- An experiment has been conducted to evaluate the effects of prioritizing use cases. All reviewers used the same use cases, but half of them used a prioritized list and the other half used a non-prioritized list. The reviewers inspected a design document. It was found that the prioritization affected the result of the inspection and that more important faults, from users' point of view, are detected in this way [9].
- An experiment has been conducted to compare UBR against checklist-based reading (CBR) for design specifications. It was found that the UBR technique was more efficient as well as effective than the CBR technique [10].

1.3 Research questions

The research questions of the two previously conducted experiments were whether prioritization affects the inspection results and which of UBR and CBR is most efficient and effective. The following two research questions are evaluated in this chapter:

- RQ1: Is there a difference in efficiency between UBR and CBR for inspection of requirement specifications? This difference may be in finding faults in general or in detecting the most severe faults.
- RQ2: How much information should be provided in the use cases of UBR when design inspections are performed?

The two research questions are researched in two different studies. RQ1 is investigated in experiment 1, and RQ2 is investigated in experiment 2. Both studies were carried out with students as subjects, and in both studies the inspected material came from the same software system. However, the two experiments were not carried out with the same students.

2 Starting Scenario

Inspection is a structured method to review software documents and is widely accepted as a cost-effective technique to improve the quality of the software. Software inspections are carried out in order to find faults at an early stage of the software development process. The general steps of an inspection process are as described in Section 1.1.

Code and design specifications have, as described above, long been exposed to this kind of review [1], but also requirements specifications have been the object of inspection [4]. As a baseline, it could be said that inspections are carried out in almost all steps of software development and in most kind of projects. In many cases, inspections are carried out as described above.

During the individual inspection, reading techniques have been introduced to guide reviewers. The common purpose of the reading techniques is to help reviewers to detect more faults (effectiveness), and to detect more faults in less time (efficiency). Several reading techniques have been proposed, for example, checklist-based reading [1], defect-based reading [4], perspective-based reading [5] and usage-based reading

[9]. The work on these techniques is summarized by Thelin et al. [10]. Checklist-based reading is still regarded as the most commonly used reading technique in industry [11].

3 Method

The two experiments were carried out as *controlled experiments*, i.e. the experimenters had control over the execution of the experiments, for example, they were able to decide which subjects that should use which treatment in the inspections that were carried out.

3.1 Design

3.1.1 Experiment 1: Requirements inspection

The purpose of experiment 1 is to evaluate UBR versus CBR for requirements inspections. The independent and dependent variables are defined as follows:

- Independent Variable – the reading technique, UBR or CBR.
- Dependent Variables – inspection time, meeting time, and faults. These measurements are used to calculate the effort, efficiency and effectiveness.

In the experiment, the subjects were divided into two groups. One group used UBR in the inspection, and the other group used CBR.

The null and alternative hypotheses were defined as follows:

- H0_eff: There is no difference between UBR and CBR with respect to effectiveness (number of faults found per reviewer)
- Ha_eff: There is a difference between UBR and CBR with respect to effectiveness
- H0_rate: There is no difference between UBR and CBR with respect to efficiency (number of faults found per reviewer and hour)
- Ha_rate: There is a difference between UBR and CBR with respect to efficiency

For statistical comparison of the results, a nonparametric test (Mann-Whitney U-test) was used. P-values less than 0.05 were regarded as significant.

3.1.2 Experiment 2: Design inspection

The purpose of experiment 2 is to investigate the amount of information needed for UBR inspections. This is evaluated for design inspections. This is conducted by comparing one group of reviewers that utilize developed use cases with another group of reviewers that has to develop use cases during inspection. The latter group was provided with the title and purpose of the use cases but not the tasks. Tasks of a use case are step-by-step instructions for one use case, called task notation in [12]. The independent, controlled and dependent variables are defined for the experiment. These variables are:

- Independent Variable – the reading technique used. The reading technique used was either utilizing (purpose and tasks pre-developed) use cases or developing (only purpose pre-developed) use cases. These treatments were used in two different places, in campus Helsingborg at Lund University and in Ronneby at Blekinge Institute of Technology. The abbreviations used 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.
- Controlled Variable – the experience of the reviewers. This is measured on an ordinal scale. The reviewers filled in a questionnaire with seven questions.
- Dependent Variables – preparation time, inspection time and faults. These measurements are used to calculate the effort, efficiency and effectiveness.

The hypotheses of the experiment were 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 point of view.

- H0_eff – There is no difference in efficiency (i.e. found faults per hour) between reviewers utilizing pre-developed use cases (Util) and reviewers who develop use cases (Dev).
- Ha_eff – There is a difference in efficiency between reviewers utilizing pre-developed use cases (Util) and reviewers who develop use cases (Dev).
- H0_rate – There is no difference in effectiveness between reviewers utilizing pre-developed use cases (Util) and reviewers who develop use cases (Dev).
- Ha_rate – There is a difference in effectiveness (i.e. rate of faults found) between reviewers utilizing pre-developed use cases (Util) and reviewers who develop use cases (Dev).

For statistical comparison of the results, a nonparametric test (Mann-Whitney U-test) was used. P-values less than 0.05 were regarded as significant.

3.2 Subjects

In both experiments, students were used as subjects. In experiment 1, the students were in their 3:rd or 4:th year of a Masters education. In experiment 2, the students were in their 3:rd year at a Bachelors education and in their 4:th year at a Masters education.

In study 1, there were 29 subjects, and the prior experience of them is shown in Figure 1. The dots represent the individual sum of scores within five experience areas (programming, requirement inspections, use cases, taxi systems and using taxis). The horizontal lines represent the median values. There was some difference between the experience of individuals, but there was no significant difference between the groups. The prior experiences were collected through a questionnaire after the inspection meeting. Hence, no controlled variable was used since the experience of the subjects was checked after the experiment.

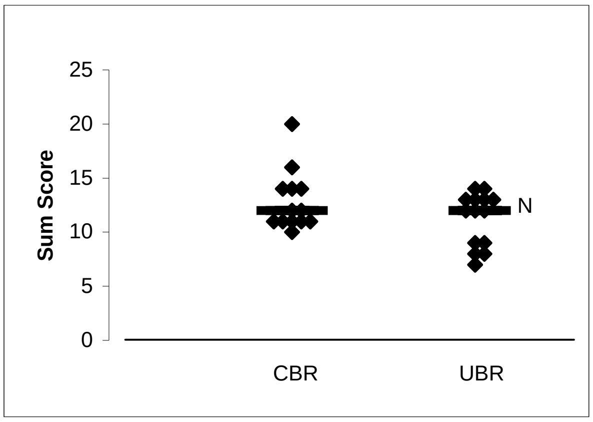

FIGURE 1. Previous experience of the students that acted as reviewers in the experiment.

In experiment 2, there were 82 subjects, 34 Bachelor students from Lund University (Hbg) and 48 Master students from Blekinge Institute of Technology (Rb). The experiment was a mandatory part of two 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 material. 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 main difference between the students in Hbg and Rb can be referred to their education, domain knowledge, and industrial experience.

3.3 Apparatus/material

In both experiments, the inspected documents came from the same system, a taxi management system developed by the Department of Communication Systems, Lund University [9]. The system is a simplified version of a system for managing a number of taxi-cars with drivers, and a central node with an operator. The system consists of the following parts (see Figure 2):

- Taxi-component: a computer and a communication device in every taxi. The taxi drivers operate this component.
- Central node: a central computer, which is operated by an operator.
- Communication link: provides communication between the central system and the taxi-components.

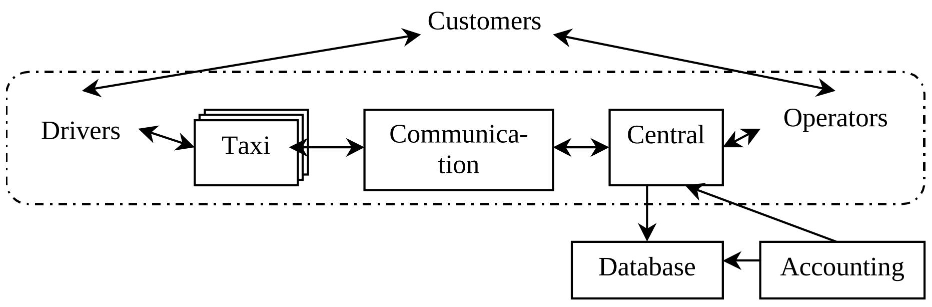

FIGURE 2. The taxi management system. The boxes represent software modules. The software for the database and accounting system is not implemented in the first version.

The following artifacts are available for the taxi management system:

- Textual requirements: The requirements are written in a feature style notation [12] using natural language (English). The document includes a glossary, a state chart, and a context diagram (totally about 10 pages).
- Use cases for the system: The use case document (10 pages) consists of 24 use cases in prioritized order with the most important use case first.
- Design document
- Checklists for design inspection (not used in the experiments presented here).
- Checklist for requirements inspections: The checklist for requirements inspections is rather short and includes checks for: correctness, completeness, consistency, unambiguous, realistic, and verifiable.
- Code and test cases (not used in the experiments presented here).

3.4 Procedure

In this section, the procedure for both experiments is presented. The following fault classification was used in both experiments:

- A-faults: The most severe faults in critical functions or frequently used functions. Such faults were considered most important for a user of the system.
- B-faults: Moderately severe faults. These faults were found in not so important functions (from a user's point of view) or in not frequently used functions.
- C-faults: Faults that have only a minor impact on the functions.

3.4.1 Experiment 1: Requirements inspection

Experiment 1 was carried out as shown in Table 1. In step 1, 14 faults were seeded into the requirements document, in which there could be a number of other faults. The seeded faults were mostly faults that previously had been removed from earlier versions of the requirements specification.

TABLE 1. Summary of experiment 1.

Step	**Description**	**Carried out by**
1	Fault seeding	Researchers
2	Division into inspection groups	Subjects
3	Start-up meeting	Researchers, subjects
4	Individual inspection	Subjects
5	Inspection meeting	Subjects (inspection groups)
6	Fault classification	Researchers
7	Analysis	Researchers

Two treatments were chosen for the study, UBR and CBR, where CBR was used as a baseline. The students assigned themselves to six inspection groups, i.e. they selected whom to perform the inspection meeting with. Each group consisted of about five people. Hence, randomization was performed on the group level, not on the individual level, i.e. the researchers randomly assigned the treatments to the groups. In step 3 of the experiment the students participated in a start-up meeting. At this meeting they got their individual package of documents to be inspected and forms on paper to fill in. The package included instructions, the requirements specification and use cases or a checklist. They also got detailed instructions for the remaining steps in the process. The two treatments (CBR or UBR) were randomly assigned to three inspection groups. 14 students were assigned to use CBR during inspection and 15 were assigned to use UBR. Then, in step 4, the students inspected the requirements specification individually at home, using the assigned reading techniques. They should use maximally about two hours. In step 5, each inspection group conducted a meeting where they discussed the faults found and produced a common list of faults that they could agree on. After the inspection the students were asked to grade their previous experience (5 areas with 7 questions, each with score 1 to 5) and reveal their educational background. These answers were used during the interpretation of the results of the study. For ethical reasons the students did not write their names on any form. Instead each group got unique identification numbers to distribute within the group in order to connect a form with a group and a treatment.

All documents and forms were written in English. No students have English as their native language.

Before analyzing the result, the issues found by the reviewers were discussed and classified (Step 7). First, a list of all issues found individually was produced. If there were more than one issue referring to the same original fault in the document they were regarded as one fault only. In case the reported faults were not judged as true faults (after precise consideration by the authors of this paper), they were discarded. Finally, all faults in the final list were classified (by the authors of this paper) according to their severity for the function of the final software product. The fault classification was made by the researchers to achieve a common understanding of the faults.

Although the results that are presented in this chapter are derived by the authors, the data produced during the experiment were also analyzed by the students. The objective of this was to teach the students how to carry out empirical studies. This teaching methodology is further presented in [13].

3.4.2 Experiment 2

Prior to experiment 2, the students were asked to fill in an experience questionnaire used for the controlled variable. Using the controlled variable, the students were divided into three groups and then randomized within each group. The three groups consisted of three experience levels (high, medium and low). For each of these levels, the students were randomly assigned to one of the reading techniques, Util or Dev. The division was carried out in Hbg and Rb, separately.

The faults were classified in A, B and C, as in experiment 1. The design document included 38 faults, where 30 were real faults injected during the development of the taxi management system; 8 faults were seeded into the design document by the person who developed the system.

The experiment was conducted during three days, both in Hbg and Rb. However, since the students in Hbg have 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 of the taxi management system and the training part were only performed for the students in Rb. The training part included a brief presentation of the reading technique (either Util or Dev) and a small pilot example where the subjects used the reading technique assigned to each group.

The inspection was carried out in a classroom, where at least one of the researchers was present all the time. The subjects received an inspection package containing instructions, a requirements document, a design document, a use case document and an inspection record. During the inspection, the subjects firstly read the documents briefly (about 20 minutes), and then inspected the design document. The inspection was performed individually at maximum 3 hours and 45 minutes.

All documents and forms were written in English. No students have English as their native language.

The analysis was performed by one of the researchers. In this step, false positives were removed and the data were prepared for the statistical analysis. After the first analysis session, another researcher checked the inspection records and the data.

After the experiment had been analyzed, a debriefing session was held with the subjects. The session included a presentation and a discussion of the results of the experiment.

4 Results

4.1 Experiment 1: Requirements inspections

A first list of 89 possible faults reported individually from reviewers was produced. At the first analysis of this list the number of faults were reduced due to one of the following reasons.

- Minor language remarks/criticism. (1 case)
- Duplicates of other faults. (11 cases)
- Conflicts only with use cases and not within the requirement document. (5 cases)
- Not considered as real faults (after careful judgment by the authors of this paper). (25 cases)

47 true faults remained and they were classified in 16 A-faults, 15 B-faults and 16 C-faults.

Below the results from the individual inspections are presented. Since there were so few inspection groups, the results for each group are not presented. Effectiveness refers to the number of faults found independent of the time consumed by the reviewers. Faults that could be most relevant for the user of the final software product were assumed to be A-faults or maybe A- and B-faults together. Figure 3 shows the effectiveness of the individual inspection considering the different classes of faults and comparing the two different reading techniques, CBR and UBR. There were no significant differences except for B-faults and C-faults. Thus, UBR-reviewers were significantly more effective than CBR-reviewers in finding B-faults but less effective in finding C-faults. No statistical significant differences could be observed in their effectiveness in finding A-faults, both A- and B-faults, or all faults (A+B+C).

The efficiency during fault detection refers to the number of faults found per time unit. Figure 4 shows the efficiency in terms of number of faults found per hour. The result shows that the CBR approach was significantly more efficient in finding faults than the UBR approach. This can be seen for A- and B-faults, as well as all faults (A+B+C). UBR-reviewers were significantly more efficient in finding B-faults. Interestingly, most UBR-reviewers found no C-faults at all.

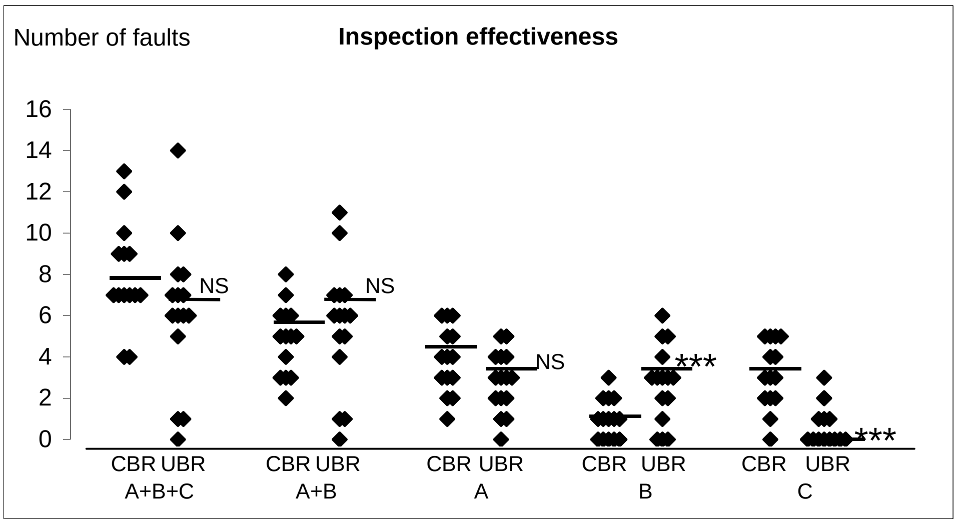

Figure 3. Individual inspection effectiveness. Individual results are shown as black squares. Horizontal lines represent median values. There were 14 reviewers using CBR and 15 reviewers using UBR. (NS= non-significant, * means p<0.05, ** means p <0.01 and *** means p<0.001)

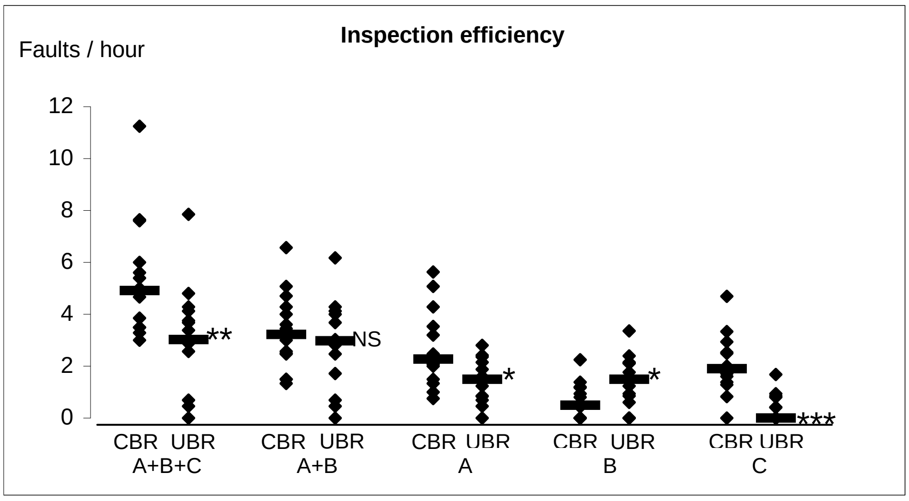

Figure 4. Individual inspection efficiency. Individual number of faults found per hour is shown as black squares. Horizontal lines represent the median value. There were 14 reviewers using CBR and 15 reviewers using UBR. (NS= non-significant, * means p<0.05, ** means p <0.01 and *** means p<0.001)

4.2 Experiment 2: Design inspection

There were 38 faults in the design document inspected. No unknown faults were found during the experiment. In this section, the efficiency and effectiveness are evaluated for both places (Hbg and Rb) together and separately.

In Figure 5, the effectiveness is shown for all faults (A+B+C), A-faults and both A- and B-faults. The two first box plots of each class of faults show Util and Dev when the reviewers from Hbg and Rb are combined. The next two plots are reviewers from Hbg, and the last two are reviewers from Rb. The same order is present in Figure 6, where the efficiency values are presented. In total, the Util groups were more efficient and effective for all faults (A+B+C), A-faults and both A- and B-faults. Furthermore, the box plots show that there is larger difference between groups in Hbg than in Rb.

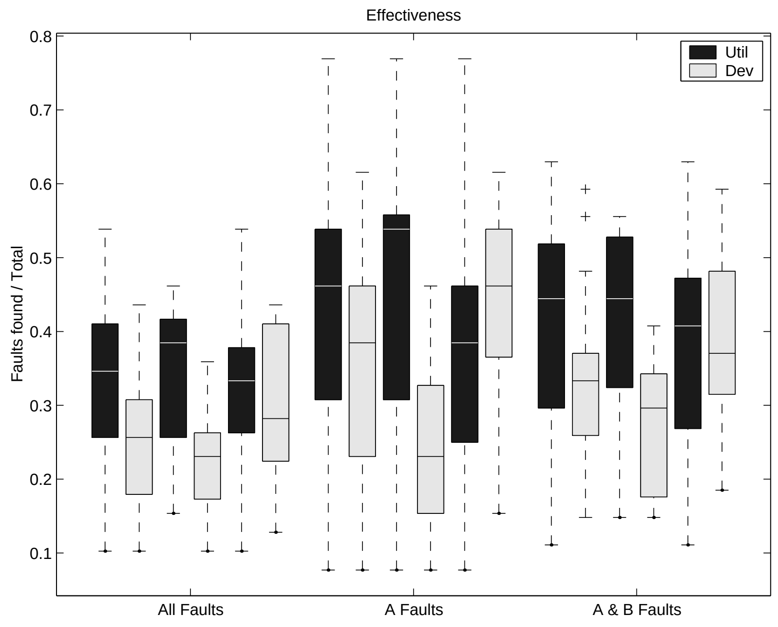

Figure 5. Individual inspection effectiveness for all faults (A+B+C), A-faults and both A- and B faults. The box plot order for each class of faults is Util (Hbg+Rb), Dev (Hbg+Rb), Util (Hbg), Dev (Hbg), Util (Rb), Dev (Rb).

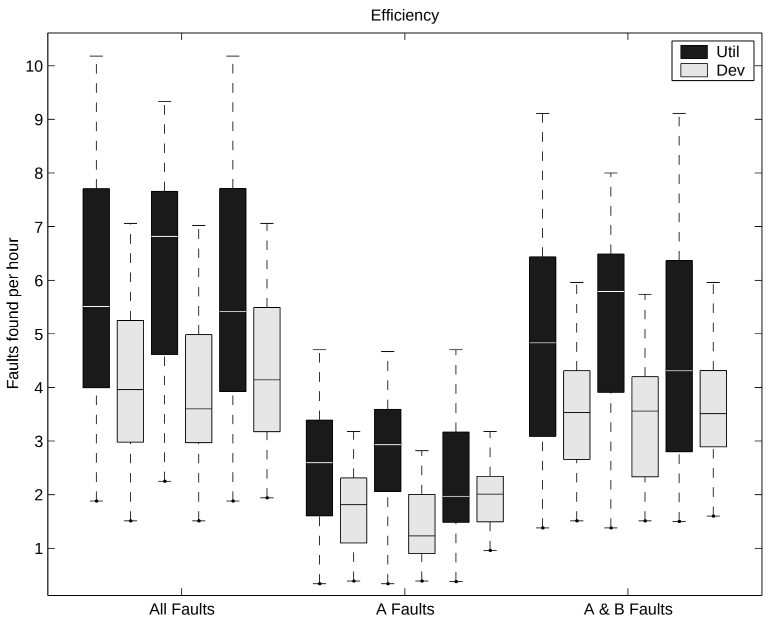

Figure 6. Individual inspection effeciency for all faults (A+B+C), A-faults and both A- and B-faults. The box plot order for each class of faults is Util (Hbg+Rb), Dev (Hbg+Rb), Util (Hbg), Dev (Hbg), Util (Rb), Dev (Rb).

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 A-faults, Dev-Rb was more effective than Util-Rb.

The p-values of the significance tests for efficiency and effectiveness are presented in Table 2. These show that the efficiency and effectiveness are significantly higher for Util than for Dev in Hbg, but not in Rb.

Regarding efficiency in Hbg, significant differences occur for all faults, A-faults and both A- and 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 A-faults and both A- and B-faults.

Regarding effectiveness in Hbg, the Mann-Whitney test shows that there is a significant difference for all faults, A-faults and both A- and 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 one between Util-Rb and Dev-Rb. This may depend on two factors; one, more time was 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, i.e. the students in Rb are better in creating use cases.

Table 2. The results of the Mann-Whitney tests. (S = significant, – = non-significant)

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

5 Discussion

The rationale for investigating the hypotheses of the experiments presented in this chapter is to extend the knowledge of the reading technique UBR. UBR has been evaluated for design inspections in two previous experiments [9] [10], and is here evaluated for requirements inspections and further evaluated for design inspections.

The following research questions were addressed in the experiments and are further discussed in this section.

- RQ1: Is there a difference in efficiency between UBR and CBR for inspection of requirement specifications?
- RQ2: How much information should be provided in the use cases of UBR when design inspections are performed?

The requirements inspection experiment shows that UBR was neither more efficient nor more effective than using CBR. The purpose of UBR is to get the reviewers better focused on the most important parts of the document, and this to detect the most critical faults from a user's point of view. However, the faults that according to the initial hypothesis were supposed to have the greatest impact on the user were best detected by CBR-reviewers. One explanation could be that the UBR approach was too heavy and exacting for the reviewers when no training session was carried out. Most of the critical faults were missed requirements or conflicts within the requirements document. To detect these kinds of faults the reviewers may need to compare relatively large parts of the document, which requires more domain knowledge. It may be easier when the reviewers just need to read the requirements document. It should be noted that this study is only one single study, and that further experimentation is needed. Opposite to this result, Thelin et al. [10] found that UBR was more efficient and effective than CBR for inspection of design documents. A difference between these studies is that in [10], the subjects were more controlled, i.e. the whole experiment was performed in a classroom.

The conclusion of the design experiment is that it 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 purpose of the experiment was to investigate whether UBR can be used without developing the use cases prior to the inspection session. Utilizing pre-developed use cases (Util) leads to reviewers becoming 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. 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 were differently introduced to the methods at the places depending on their experiences and courses taken before the experiment. Another explanation may be that the students in Hbg were less experienced. The students in Hbg were 3rd year Bachelor students and in Rb they were 4th 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.

In summary, the experimentation with UBR has shown that it is an effective and efficient reading technique for software inspections. However, there still remain questions to be further researched, for example, the experiments need to be replicated, both using design and requirements as well as code documents.

It is important to consider the validity of the results of the experiments. The following threats have been found:

- In experiment 1, the randomization of subjects was performed on the group level and not on the individual level. This means that every reviewer of a group used the same treatment (reading technique). The students themselves selected the members of the groups. After the experiment a post-test measuring the experience of the reviewers was performed. The result of this test shows that there was no significant difference between the reviewers of the different treatments. In experiment 2, the experiences of the subjects were measured, on an individual level, prior to the experiments and were used in the controlled variable.
- In experiment 1, the individual inspection was conducted at home. The validity is dependent on that the students honestly report the actual time consumption for the inspection. On the other hand it is assumed that most individual inspections in the industry are conducted very informally, similar to the conditions in the present study. In experiment 2, the individual inspection was carried out in a classroom setting.
- Inspected documents and written instructions are in English, and not in the reviewers' native language. In both experiments, it could have been a burden for some of the students. Though, they could use their own native language when they described the faults in the form. Many Swedish companies write their software documents in English.
- In experiment 1, there was no training period for the students before the real inspection. They might have been too inexperienced to make a real inspection especially using UBR as a reading technique. In experiment 2, there was a training session prior to the experiment session.
- The classification of faults according to their severity is very subjective. However, faults were discussed by all the authors of this paper before the final decision.

These threats should be taken into account when conclusions are drawn. Whether the findings could be generalized to be valid for requirements inspections also in industry is dependent of the generalisability. In this study students were subjects. One can argue that the students' experience is too low compared with practitioners in the industry. However, Porter and Votta [14] reported that although students might have lower performance the result of an experiment with students would be the same as if professionals in the industry have been subjects in the study. In [15], it is also shown that it, under certain conditions, is not a large difference between students and professionals. This threat should also be taken into account when conclusions are drawn based on the result.

6 Lessons learned

Four experiments on UBR have been conducted, three on design documents and one on requirements document. The three design experiments show positive results in favor for UBR. However, more experiments are needed in order to draw further conclusions about UBR for requirements inspections.

The lesson learned from these studies is that UBR is an efficient and effective reading technique when it is applied to design specifications. Software organizations that utilize use cases during development can preferable prioritize and provide them as input to the reviewers. If use cases not are used, the title and purpose need to be developed before they are used as inspection aid during the individual inspection.

On the other hand, when UBR is used for requirements inspection, it is not possible to conclude whether UBR or CBR should be used. Instead it shows results that are not in line with earlier studies, which makes this study important. The study emphasizes the need for further experimentation in the area.

7 References

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

[2] Aurum, A., Petersson, H., Wohlin, C., "State-of-the-Art: Software Inspections after 25 Years", Software Testing, Verification and Reliability, 12(3):133-154, 2002.

[3] Fagan, M. E., "Advances in Software Inspections", IEEE Transactions on Software Engineering, 12(7):744-751, 1986.

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

[5] Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M. V., "The Empirical Investigation of Perspective-Based

Reading”, Empirical Software Engineering: An International Journal, 1(2):133-164, 1996.

[6] Olofsson, M., Wennberg, M., “Statistical Usage Inspection”, Master’s Thesis, Department of Communication Systems, Lund University, CODEN:LUTEDX(TETS-5244)1-81/1996&local. 9, 1996.

[7] Thelin, T., Runeson, P. Wohlin, C., “Prioritized Use Cases as a Vehicle for Software Inspections”, to appear in *IEEE Software*.

[8] Ackerman, A. F., Buchwald, L. S. and Lewski, F. H., “Software Inspections: An Effective Verification Process”, *IEEE Software*, 6(3):31-36, 1989.

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

[10] Thelin, T., Runeson, P. Wohlin, C., “An Experimental Comparison of Usage-Based and Checklist-Based Reading”, to appear in *IEEE Transactions on Software Engineering*.

[11] Laitenberger, O., DeBaud, J.M., “An Encompassing Life Cycle Centric Survey of Software Inspection”, *Journal of Systems and Software*, 50(1), pp. 5-31, 2000.

[12] Laueson, S., “Software Requirements – Styles and Techniques” Addison-Wesley, Pearson Education Limited, UK, 2002.

[13] Höst, M., ”Introducing Empirical Software Engineering Methods in Education”, Proceedings of Conference on Software Engineering Education and Training, pp. 170-179, 2002.

[14] Porter, A, Votta, L., “Comparing Detection Methods for Software Requirements Inspection: A Replication Using Professional Subjects”, Empirical Software Engineering, 3(4), pp. 355-380, 1998.

[15] Höst, M., Regnell, B., Wohlin, C. “Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment”, Empirical Software Engineering, 5(3), pp. 201-214, 2000.