Empirical Software Engineering, 9, 77–110, 2004.

Evaluation of Usage-Based Reading—Conclusions after Three Experiments

THOMAS THELIN thomast@telecom.lth.se
Department of Communication Systems, Lund University, Box 118, SE-221 00 Lund, Sweden

PER RUNESON perr@telecom.lth.se
Department of Communication Systems, Lund University, Box 118, SE-221 00 Lund, Sweden

CLAES WOHLIN claes.wohlin@bth.se
Department of Software Eng. and Computer Science, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Sweden

THOMAS OLSSON thomas.olsson@iese.fhg.de
Fraunhofer IESE, Sauerwiesen 6, 67661 Kaiserslautern, Germany

CARINA ANDERSSON carinaa@telecom.lth.se
Department of Communication Systems, Lund University, Box 118, SE-221 00 Lund, Sweden

Editor: Ross Jeffery

Abstract. Software inspections have been introduced in software engineering in order to detect faults before testing is performed. Reading techniques provide reviewers in software inspections with guidelines on how they should check the documents under inspection. Several reading techniques with different purposes have been introduced and empirically evaluated. In this paper, we describe a reading technique with the special aim to detect faults that are severe from a user's point of view. The reading technique is named usage-based reading (UBR) and it can be used to inspect all software artefacts. In the series of experiments, a high-level design document is used. The main focus of the paper is on the third experiment, which investigates the information needed for UBR in the individual preparation and the meeting of software inspections. Hence, the paper discusses (1) the series of three experiments of UBR, (2) the individual preparation of the third experiment, and (3) the meeting part of the third experiment. For each of these three parts, results are produced. The main results are (1) UBR is an efficient and effective reading technique that can be used for user-focused software inspections, (2) UBR is more efficient and effective if the information used for UBR is developed prior to, instead of during the individual preparation, and (3) the meeting affects the UBR inspection in terms of increased effectiveness and decreased efficiency. In summary, the empirical evidence shows that UBR is an efficient and effective reading technique to be used by software organizations that produce software for which the user perceived quality is important.

Keywords: Empirical study, reading technique, software inspection, usage-based reading.

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 (Aurum et al., 2002). Researchers have over the years proposed several different

ways of improving software inspections. The improvements include new inspection processes, for example, *n*-fold inspection (Martin and Tsai, 1990) and phased inspections (Knight and Myers, 1993). Improvements also include changes to the different steps in the inspection processes, for example, new reading techniques (Basili et al., 1996) and whether an inspection meeting is needed or not (Votta, 1993). Other improvements include support to the inspection process, for example, fault content estimations (Petersson et al., 2004). The objectives of this paper are in general to contribute to the improvement of reading techniques and, specifically, to study the new reading technique, usage-based reading (UBR) (Thelin et al., 2003b).

The user perspective in software development is acknowledged and valued in different methods including for example use cases in object-orientation (Jacobson et al., 1992) and operational profile testing (Musa, 1998). However, the user perspective can also be applied to 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 UBR (Thelin et al., 2003b) and stems from an idea by Wohlin and Ohlsson, which is presented in Olofsson and Wennberg (1996). Since then, UBR has been extended and evaluated in two main studies (Thelin et al., 2001; Thelin et al., 2003a).

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 UBR. The first experiment focused on comparing use case driven inspections with prioritized use cases versus randomly ordered use cases. After having found out that the group using prioritized use cases, which constitutes a key part in UBR, performed significantly better, UBR was compared with checklist-based reading (CBR). UBR was found to be significantly better than using a checklist. The third experiment investigates 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 UBR as part of the inspection process, i.e., a meeting is also held in the third study. It is concluded that UBR is an effective and efficient reading technique and that it can be integrated into the software inspection process. In summary, the paper reports on and draws conclusions about (1) a series of three UBR experiment, (2) the individual preparation of UBR in the third experiment, and (3) the inspection meeting of the third experiment.

The paper is outlined as follows. UBR is described in Section 2, together with an overview of the series of experiments. The main focus of this paper is on the third experiment, which is discussed in detail in Sections 2.4 to 8. The artefacts for the experiment are presented in Section 3. In Section 4, the planning of the experiment can be found. The operation of the experiment is discussed in Section 5 and the analysis is presented in Sections 6 and 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 (1976)) 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 (DBR) (Porter et al., 1995) and perspective-based reading (PBR) (Basili et al., 1996). UBR is one 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) (Saaty and Vargas, 2001)) 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 (Musa, 1998). 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. In this paper, it is investigated how much information that is needed in order to apply UBR, see Section 2.4.

There are some other reading techniques that utilize use cases during fault searching. Among these are traceability-based reading (TBR) (Travassos et al., 1999) and the user perspective in PBR (Basili et al., 1996). 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. Dunsmore et al. (2002) compare a use case approach with CBR and structured reading for inspections of code. 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. Current research of reading techniques in software inspections are summarized in Thelin et al. (2003a).

2.1. 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 programs in their third or fourth years of studies.

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.

The sequence of experiments is summarized in Table 1. The first and second experiments are presented in Sections 2.2 and 2.3, respectively, and the third experiment is presented in Section 2.4 and onwards.

2.2. First Experiment

The first experiment, which was aimed at investigating the basic principle of UBR, was launched during the fall semester of 2000. Twenty-seven students of their third year of the software engineering Bachelor's program 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 3.2 for fault classification). Half of the subjects was given use cases that were prioritized with respect to the impact on the intended users of the system. The other half of the subjects was given the same use cases, but in a random order. The experiment is presented in more detail in Thelin et al. (2001).

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?

The efficiency and effectiveness values are presented in Table 9 page 105 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 ($p = 0.004$). 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.

2.3. Second Experiment

The purpose of the second experiment was to compare UBR to CBR. CBR is some kind of industry practice (Laitenberger and DeBaud, 2000), and was used as a baseline to which UBR was compared. The experiment was launched during the spring semester of 2001. Twenty-three students of their fourth year of the software engineering Master's program at Blekinge Institute of Technology inspected the same high-level design (as in the first experiment), although the design document 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 was taught the UBR method and was given prioritized use cases. The other half of the subjects was taught the CBR method and was given a checklist, where the items were sorted in a significance order. The experiment is presented in more detail in Thelin et al. (2003a).

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 ($p = 0.042$, 35% more faults per hour). This was also true for class A faults ($p = 0.013$, 95% more class A faults per hour) and class A&B faults ($p = 0.016$, 70% 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.

2.4. 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 3.1), 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 (Sections 3 to 7), the third experiment is described in detail. Then, in Sections 8 and 9, the results of the third experiment are discussed together with a discussion of the series of experiments.

3. 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 them is provided in Thelin et al. (2001) and Thelin et al. (2003a).

3.1. 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) (ITU-T Z.100, 1993). 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 contain 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 (Lauesen, 2002). 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 (Saaty and Vargas, 2001). 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) (ITU-T Z.120, 1996), which show signaling between the modules for two different cases.

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 sec.
4. The channel is open until the central resets the alarm.

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 4.2.

- 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 experiment 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.

3.2. 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.

4. Experiment Planning

4.1. 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) of Lund University and then in Ronneby (Rb) at Blekinge Institute of Technology. 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 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 program at Lund University (Runeson, 2000). 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 (Wohlin, 1997) 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.

4.2. Variables

The independent, controlled, and dependent variables are defined for the experiment. 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.

- *Independent variable*—The independent variable is the reading technique used. The reading technique used was either utilizing (purpose and tasks provided) use cases or developing (only purpose provided) use cases. These treatments were used both

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.

- *Controlled variable*—The controlled variable is the experience of the reviewers and it is measured on an ordinal scale. The reviewers were asked to fill in a questionnaire consisting of seven questions.

- *Dependent variables*—The dependent variables measured are time and faults. These are listed below.

1. Time spent on reading in the preparation phase, measured in minutes.

2. Time spent on inspection in the preparation phase, measured in minutes.

3. Time spent in meeting, measured in minutes.

4. Clock time when each fault was found, measured in minutes, from start of preparation.

5. Number of faults found by each reviewer.

6. Number of faults found by each group of reviewers in the meeting.

7. Efficiency for the preparation phase, measured as: $60 \cdot$ (Number of faults found/ Reading time + Inspection time).

8. Efficiency for the meeting phase, measured as: $60 \cdot$ (Number of faults found/ Meeting time of the reviewers in each group).

9. Effectiveness, measured as: Number of faults found/Total number of faults. This was measured for both the preparation and meeting phases.

4.3. Hypotheses

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 perspective.

The dependent variable is analyzed to evaluate the hypotheses of the experiment. The main alternative hypotheses are evaluated for all faults, class A faults and class A&B faults. The hypotheses concern efficiency, effectiveness, and fault detecting differences.

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

- 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.

The above hypotheses are investigated for the preparation phase as well as for the meeting phase.

4.4. 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 (Wohlin et al., 2000). 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} (Siegel and Castellan, 1988).

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 (1988).

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.

4.5. 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 (Wohlin et al., 2000).

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 6.1.

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.

5. Experiment Operation

The experiment was run in September 2001. The experiment was conducted during two days 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).

The second session of A 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. During this session, they were allowed, and encouraged, to ask as many questions as possible about the material and the reading technique. However, they were not allowed to know the other group's reading technique. Consequently, they were not allowed to discuss their reading technique with any person in the other group.

The inspection meeting was performed during the second day. The goal of the meeting was to compile the faults that were found in the preparation phase and to find new faults.

The package for the individual part of the experiment contained:

1. Inspection record, including:

   - A description of the fault classification scheme (severity)
   - A time log for reading and inspection time
   - A fault log. For each fault found, the reviewers logged the use case, the clock time when found, the class and severity of the fault

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 (Developing)
Day 1 (B) (13.15 p.m.–17.00 p.m.)	3 h 45 min	Preparation and Inspection Questionnaire (15 min)	
Day 2 (13.15 p.m.–16.00 p.m.)	2 h 45 min	Inspection meeting Questionnaire 2 (15 min)	

2. A requirements document

3. A design document

4. A use case document.

The instructions for the students were:

1. The textual requirements are assumed to be correct. If an inconsistency is found between the requirements and the design, the fault is in the design.

2. Log all clock times, start time of reading, start and end time of inspection, clock time when a fault is found and so forth.

3. Read through all documents briefly before starting to inspect.

4. The inspection experiment is finished either when all use cases are used or after 3 hours and 30 minutes. When finished, verify that the logged data seem to be correct and hand them in.

After each student handed in their inspection record, it was checked for errors and missing data in the record in order to get as accurate data as possible.

The package for the inspection meeting contained:

1. The individual material used in the preparation phase.

2. A meeting record (a log for common faults found, for the specific team).

3. An inspection record (a log for new faults).

In addition to the instructions from the preparation phase, the instructions in the meeting phase were:

1. Use the individual inspection record, and decide which are faults and which are false positives.

2. Detect new faults.

3. The meeting phase is finished when every fault has been discussed and logged either as a false positive or as a true fault, or after 2 hours and 30 minutes.

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.

6. Analysis of Individual Preparation

The analysis of the experiment is divided into two sections, analyzing the individual preparation (Day 1) and the inspection meeting (Day 2).

In this section, the analysis of the individual preparation is presented. The analysis concerns time spent, effectiveness and efficiency, faults found and answers of the first questionnaire.

6.1. 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.

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.

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

		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

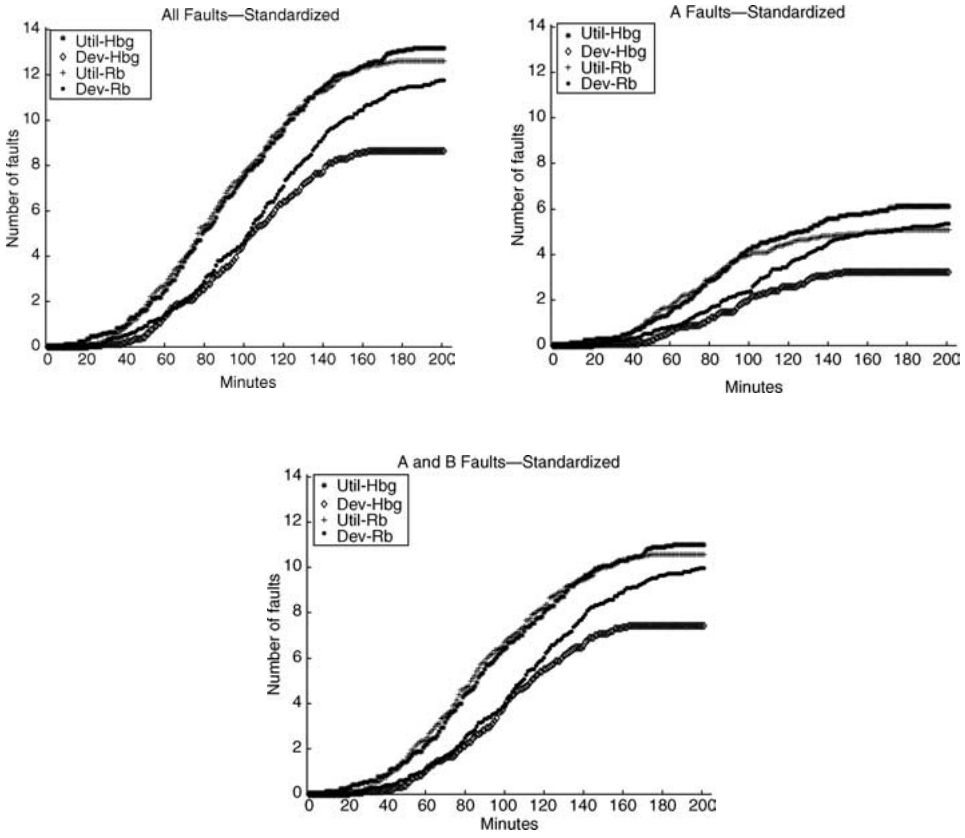

Figure 2. The figures show the faults found over time for an average reviewer. The plots are, from left to right, all faults, class A faults and class A&B faults. Preparation and inspection time are included in the figures.

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 reviewers 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.

6.2. *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 4.4.

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 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 4.4. 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.

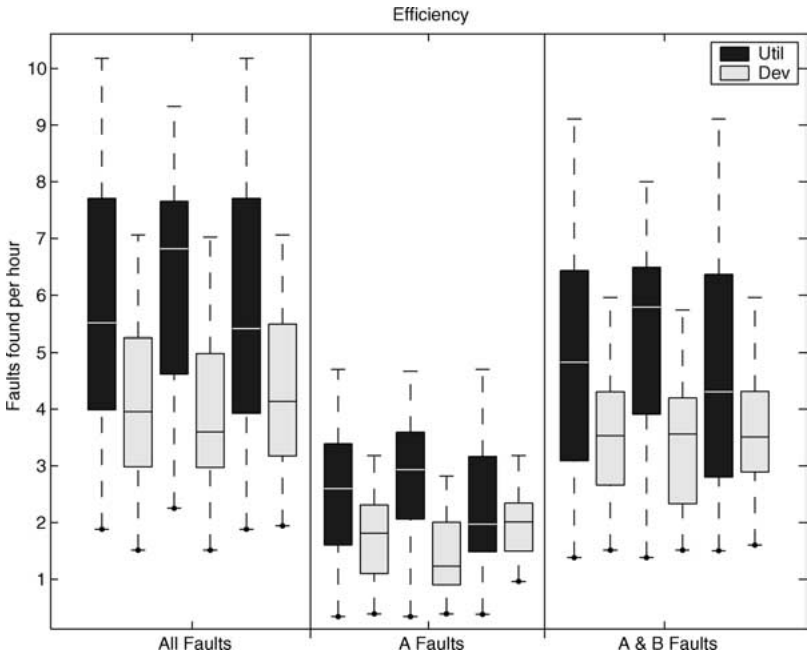

Figure 3. Efficiency for all faults, class A and class A&B faults. The box plot shows, from left to right, both groups, Hbg and Rb.

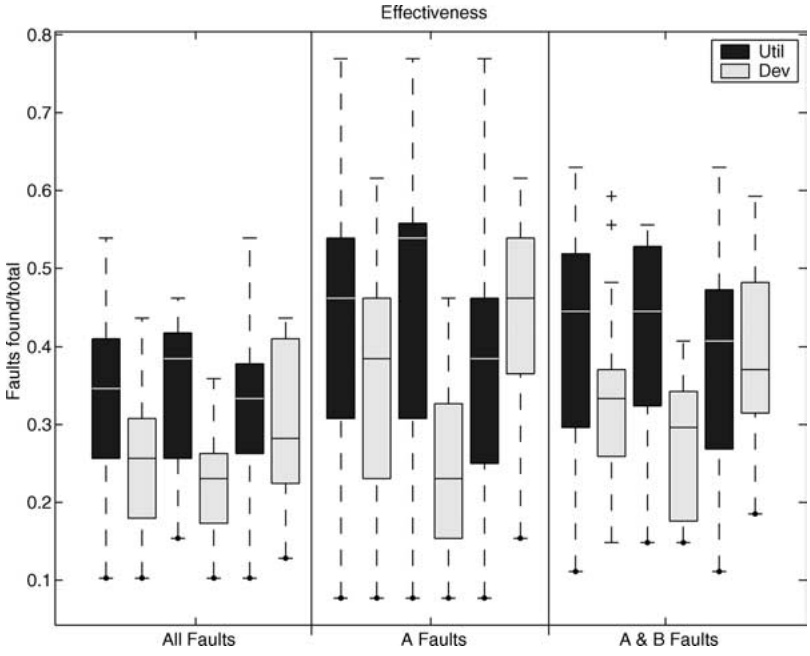

Figure 4. Effectiveness for all faults, class A and class A&B faults. The box plot shows, from left to right, both groups, Hbg and Rb.

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.

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.

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 (−)

6.3. 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

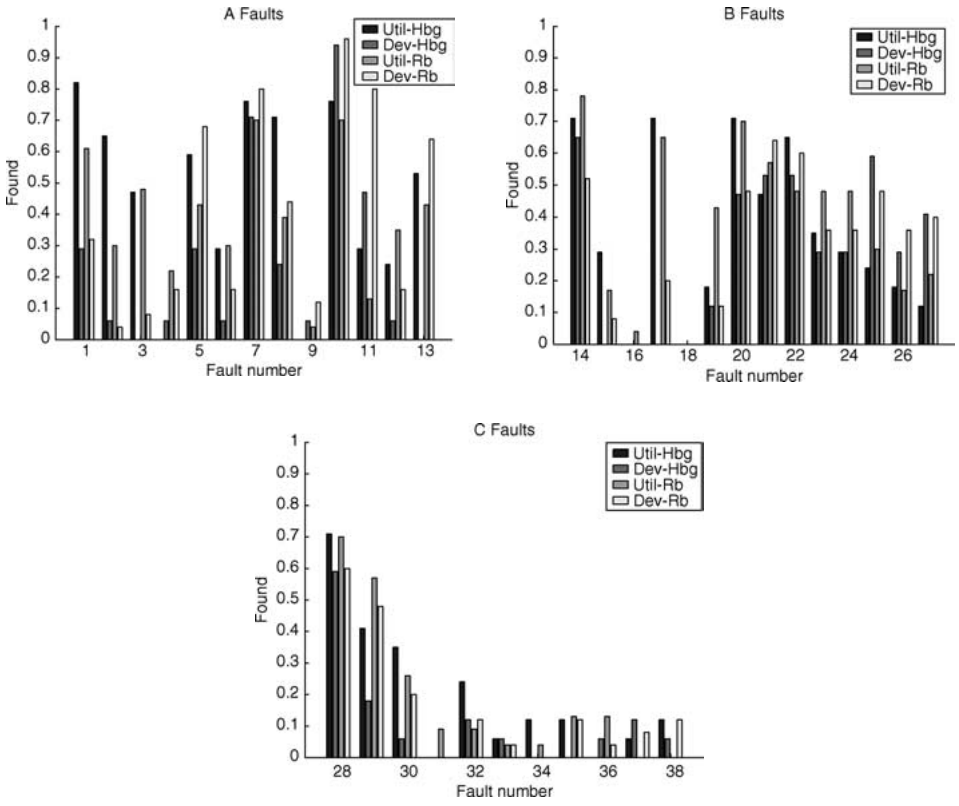

Figure 5. Bar plots of the percentage of faults found by the groups. From left to right, class A faults, class B faults and class C faults.

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

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.

6.4. 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.

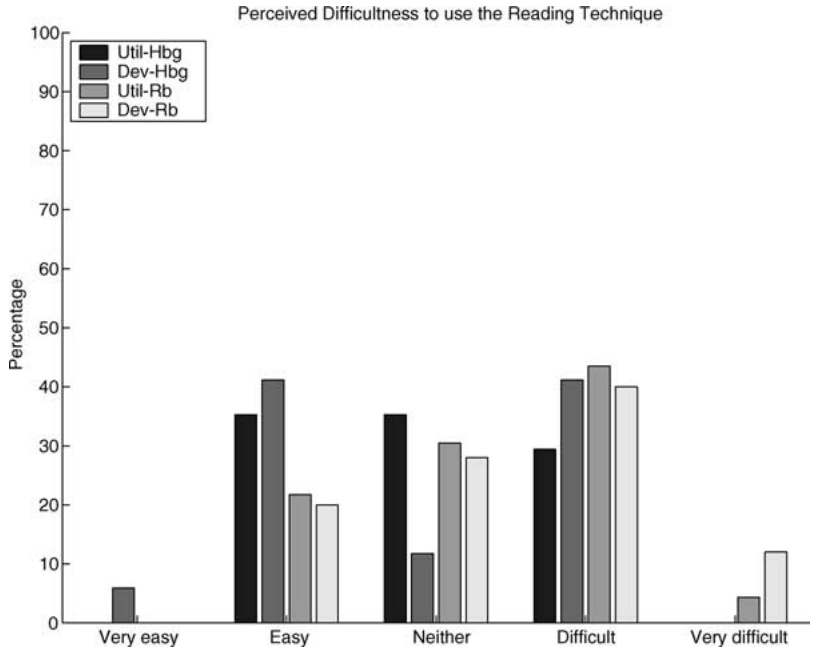

Figure 6. Bar plot of the perceived difficulty of using the reading technique.

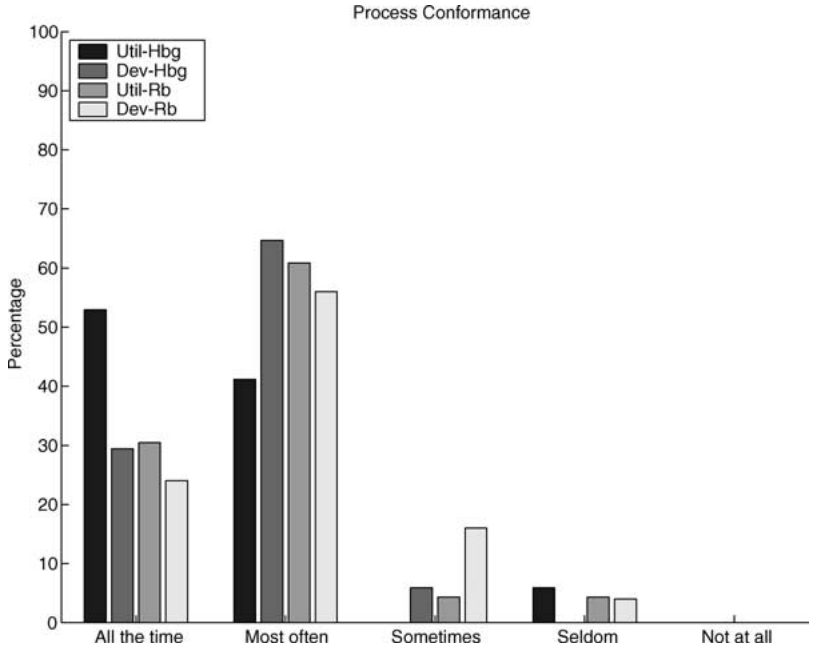

Figure 7. Bar plot of the conformance of the inspection process.

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 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.

Table 6. Mean and standard deviation values for meeting time.

	Mean (min)				Standard deviation			
	Util–Hbg	Dev–Hbg	Util–Rb	Dev–Rb	Util–Hbg	Dev–Hbg	Util–Rb	Dev–Rb
Meeting	62	51	68	58	22.7	20.9	14.4	20.4
Total	575	528	601	708	148.1	86.8	110.9	82.7

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.

7. Analysis of Meeting

After the individual preparation, the reviewers held an inspection meeting. The meeting was organized with three reviewers in each group. In this section, the analysis of the inspection meeting is presented. The analysis concerns time spent, effectiveness and efficiency, faults found, and answers of the first questionnaire.

7.1. Meeting Time

The reviewers performed the inspection meeting with the purpose of compiling the faults found, removing false positives and detecting new faults. However, no new faults were found during the meeting. The reviewers focused on the faults that already were found during the preparation. Consequently, the meeting had no positive effect considering new faults found.

In Table 6, the meeting times and the total times are shown. The total time is calculated as the teams' individual preparation times added to the meeting time multiplied with the number of reviewers in the meeting. The meeting times are in the same range and no significant difference can be observed ($p = 0.555$). However, there is a significant difference for the total times ($p = 0.028$). This is observed only between Dev–Hbg and Dev–Rb and can be explained by that the differences observed in preparation affect the total time.

7.2. Efficiency and Effectiveness

The efficiency and effectiveness for the inspection meeting are measured in the same way as the preparation phase. The total time for the preparation and meeting is used in the efficiency calculation. In Figures 8 and 9, the efficiency and effectiveness after the meeting phase are shown. The efficiency values have decreased since no new faults were found. This has led to that the differences between the treatments are smaller. However, the effectiveness values have increased from 0.25–0.35 to 0.35–

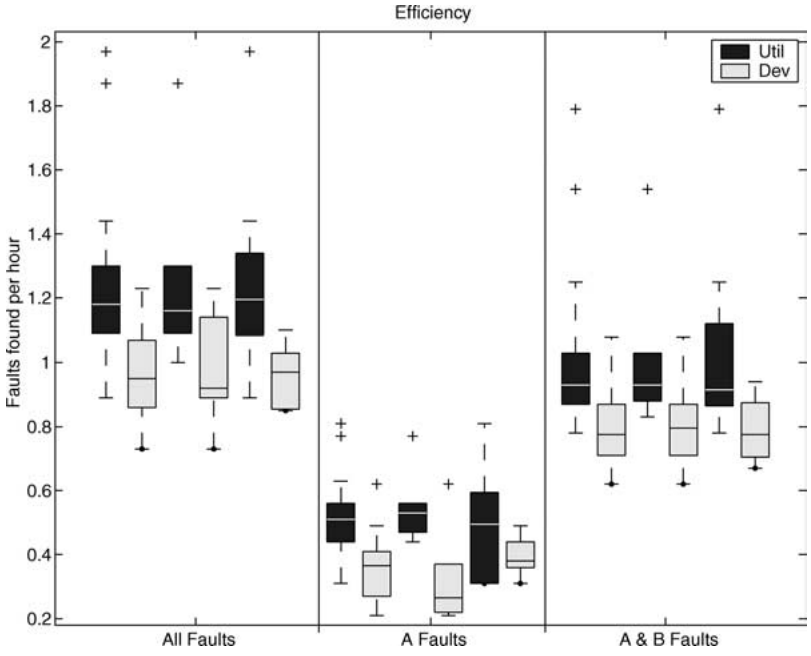

Figure 8. Efficiency for all faults, class A and class A&B faults. The box plot shows, from left to right, both groups, Hbg and Rb.

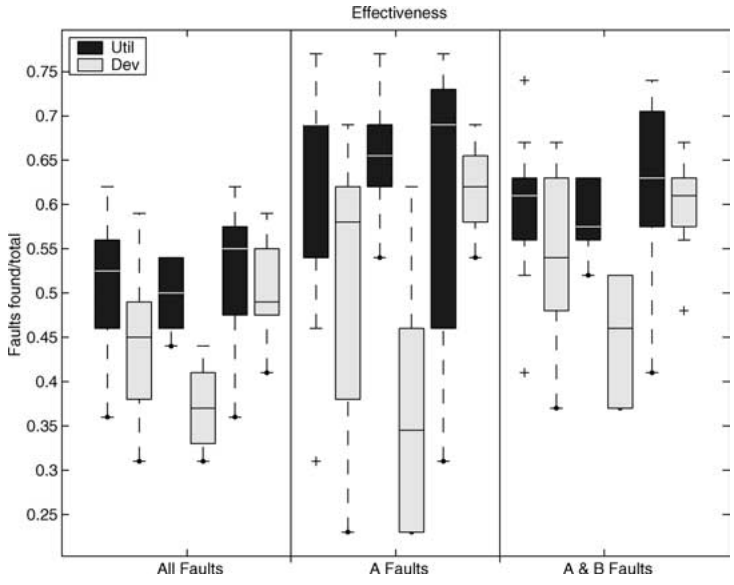

Figure 9. Effectiveness for all faults, class A and class A&B faults. The box plot shows, from left to right, both groups, Hbg and Rb.

Table 7. *P*-values for the hypotheses. Mann–Whitney U-test is used.

	Efficiency		Effectiveness	
	Hbg	Rb	Hbg	Rb
All (A + B + C)	$p = 0.078$ (−)	$p = 0.009$ (S)	$p = 0.005$ (S)	$p = 0.397$ (−)
A	$p = 0.037$ (S)	$p = 0.345$ (−)	$p = 0.010$ (S)	$p = 0.590$ (−)
A + B	$p = 0.078$ (−)	$p = 0.036$ (S)	$p = 0.006$ (S)	$p = 0.453$ (−)

0.55. This is due to the reviewers found different faults. When the reviewers compiled the faults, more unique faults were thus reported, and false positives were removed.

The hypothesis testing is performed as described in Section 4.4. The *p*-values of the significance tests for efficiency and effectiveness are presented in Table 7.

Regarding efficiency in Hbg, significant differences occur for class A faults. There are significant differences for all faults and class A&B faults in Rb.

Regarding effectiveness in Hbg, the Mann–Whitney test shows that there are significant differences for all faults, class A faults and class A&B faults. In Rb, no significant difference can be observed.

Consequently, equal to the preparation case, there is a larger difference between Util–Hbg and Dev–Hbg than between Util–Rb and Dev–Rb. However, the efficiency differences are smaller in the meeting than during the preparation phase.

7.3. Faults

When inspection meetings are conducted, the faults found during the individual preparation are combined into a new inspection record. In addition, the overlap among the faults found and false positives are identified. Bar plots of the fault distribution that each treatment found in the experiment are shown in Figure 10.

As in the individual case, the plots show that there is a higher probability to find class A and class B faults than class C faults. In order to evaluate whether the reviewers detected different faults, partial correlations are calculated. The correlations in Table 8 measure the degree of similarities among the faults found. This shows that the highest correlation is between the Util groups.

A benefit of inspection meetings is that false positives are sorted out. During the meeting, all groups reduced the false positives, i.e., the compiled inspection records

Table 8. Partial correlations between the faults found by the teams.

	Util–Hbg	Dev–Hbg	Util–Rb	Dev–Rb
Util–Hbg	1.0	0.10	**0.69**	0.07
Dev–Hbg		1.0	− 0.04	**0.61**
Util–Rb			1.0	0.33
Dev–Rb				1.0

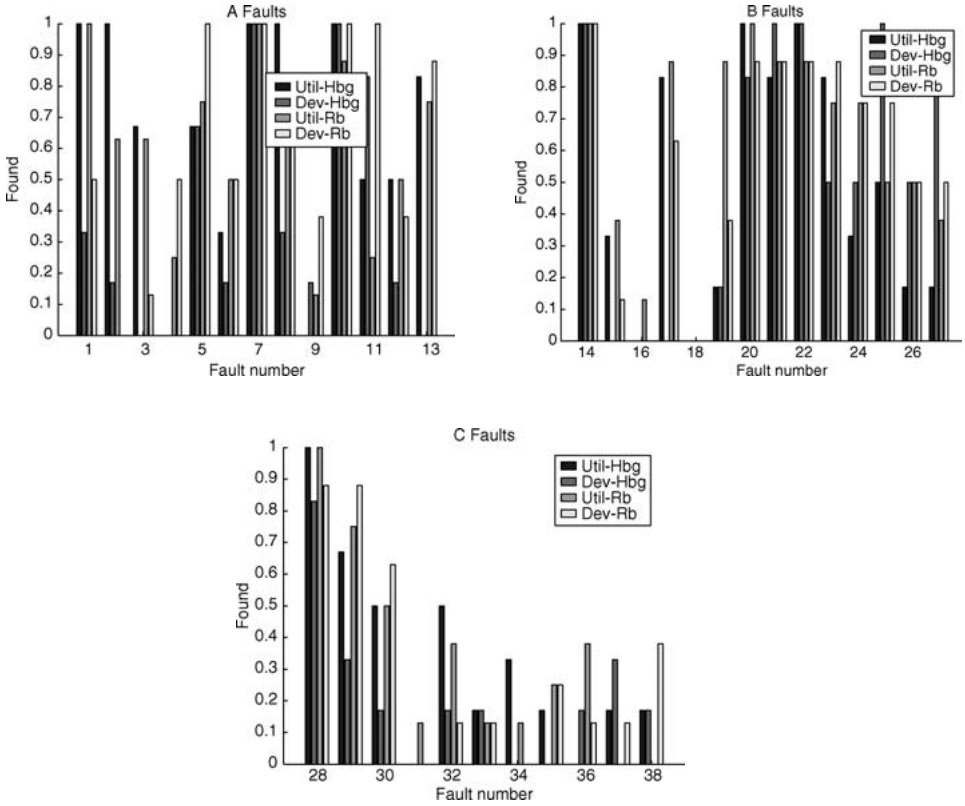

Figure 10. Bar plots of the faults found by the groups.

included less false positives than the individual inspection records for every group. On average, the meeting records after the meeting included half the number of false positives than before the meeting.

7.4. Questionnaire Data

After the meeting, the reviewers filled in a second questionnaire. The aim of the questionnaire is to explore the reviewers' opinion about inspection meetings and how difficult they thought it was to identify faults found by more than one reviewer.

Figure 11 shows the distribution of the reviewers' opinion about the inspection meetings. The bar plots show no large difference between the treatments. Most of them think the meeting is meaningful for software inspections.

Figure 12 shows how difficult the reviewers thought it was to identify the overlap among the faults found. The trend of the data seems to be that most of them thought it was easy. The data analysis shows that they identified the overlap fairly well. They

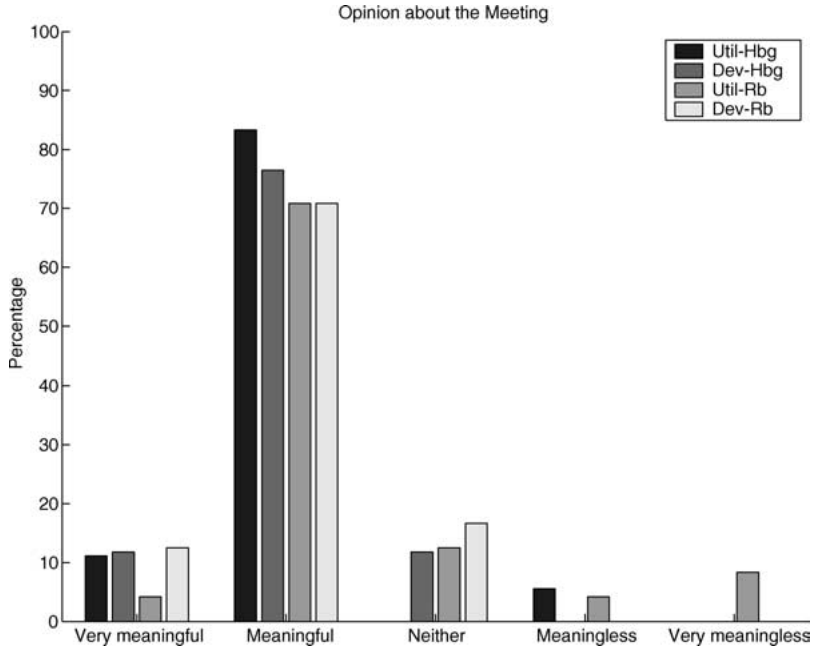

Figure 11. The reviewers' opinion about the inspection meeting.

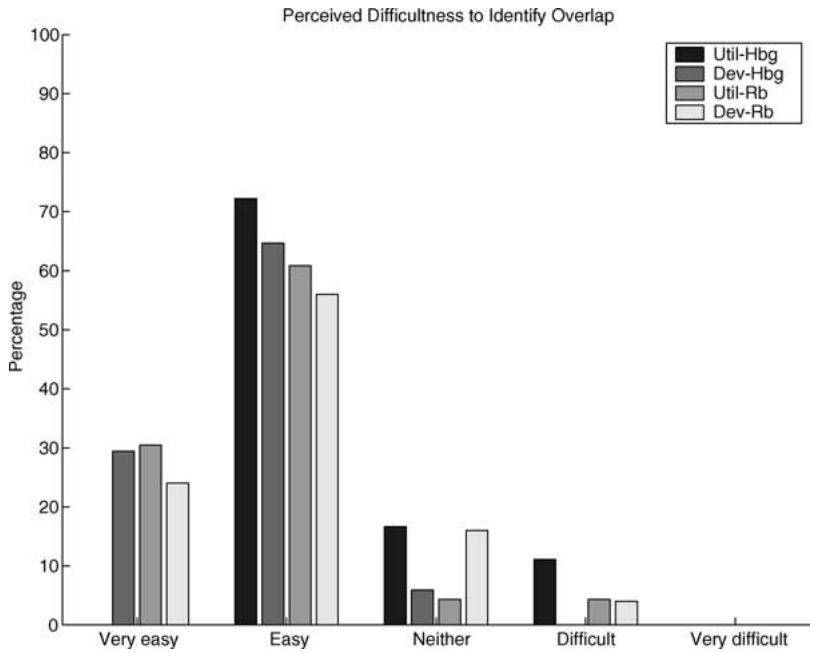

Figure 12. The reviewers' perceived difficulty to identify the overlap among the faults found.

also clustered the faults well, i.e., logged a fault as one fault instead of several. Furthermore, some false positives were removed.

Consequently, the reviewers think the inspection meeting is needed and think they performed well identifying the common faults found.

8. Discussion

The discussion is divided into two parts, individual preparation and inspection meeting. In the former, the individual preparation and the series of experiments are discussed, and in the latter, the meeting part of the third experiment is discussed.

8.1. Preparation

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. (1999) 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 program 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. (1999), when analyzing conducted DBR experiments (Porter et al., 1999). 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 (1996). 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

Table 9. 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	0.18	0.20	0.19

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 9. This indicates that the students have performed at least as well as in the other experiments.

8.2. Meeting

The hypotheses stated for the meeting part in this experiment together with the results are summarized below and in Table 10.

- 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 for class A faults. In Rb, the Util method was significantly more efficient for all faults and class A&B faults.

- H_{Rate}—For both subject groups combined, the Util method was more effective. The Util method was significantly more effective in Hbg, but not in Rb.

- H_{Fault}—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.

Table 10. Summary of the results of the hypotheses (S = significant, $\alpha = 0.05$).

	Efficiency		Effectiveness (Rate)	
	Hbg	Rb	Hbg	Rb
All (A + B + C)	Not S	S	S	Not S
A	S	Not S	S	Not S
A + B	Not S	S	S	Not S

The aim of the meeting was to detect the overlap among the faults that were found during the individual preparation and to detect new faults. The latter was, however, not fulfilled. Only few of the teams tried to detect new faults and they did not find any new ones. This led to that the efficiency decreased, i.e., more time was spent without any profit, besides reducing false positives and merging of inspection records. Hence, a quantitative evaluation of whether the meeting is beneficial or not, cannot be investigated with these experimental data. However, most of the reviewers answered, in the questionnaire, that they thought the meeting was meaningful. Furthermore, an advantage of the meeting was that false positives were sorted out.

The teams comprised three reviewers from the same inspection group. By applying an inspection meeting, the difference among the groups becomes smaller. Hence, teams may be used to decrease the variability of the performance of individual reviewers.

9. Conclusion

A series of three experiments of UBR are described in this paper, with a focus on the third one. 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 meeting of the third experiment increased the effectiveness of the faults found, but decreased the efficiency. This depended on that no new faults were detected during the meeting. However, the meeting reduced the number of false positives.

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.

Acknowledgment

The authors would like to thank the students for participating in the investigation. Thanks also to Dr. Björn Regnell and Johan Natt och Dag for help during the experiment planning. 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

Aurum, A., Petersson, H., and Wohlin, C. 2002. State-of-the-art: Software inspections after 25 years. *Software Testing, Verification and Reliability* 12(3): 133–154.

Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., and Zelkowitz, M. V. 1996. The empirical investigation of perspective-based reading. *Empirical Software Engineering: An International Journal* 1(2): 133–164.

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

Cheng, B., and Jeffery, R. 1996. Comparing inspection strategies for software requirement specifications. *Proceedings of the 8th Australian Software Engineering Conference*. pp. 203–211.

Dunsmore, A., Roper, M., and Wood, M. 2002. Further investigations into the development and evaluation of reading techniques for object-oriented code inspection. *Proceedings of the 24th International Conference on Software Engineering*. pp. 47–57.

Fagan, M. E. 1976. Design and code inspections to reduce errors in program development. *IBM System Journal* 15(3): 182–211.

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

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

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

Knight, J. C., and Myers, A. E. 1993. An improved inspection technique. *Communications of ACM* 36(11): 50–69.

Laitenberger, O., and DeBaud, J.-M. 2000. An encompassing life cycle centric survey of software inspection. *Journal of Systems and Software* 50(1): 5–31.

Lauesen, S. 2002. *Software Requirements—Styles and Techniques*. Addison-Wesley, UK.

Martin, J., and Tsai, W. T. 1990. N-fold inspection: A requirements analysis technique. *Communications of ACM* 33(2): 225–232.

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

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

Petersson, H., Thelin, T., Runeson, P., and Wohlin, C. 2004. Capture-recapture in software inspections after 10 years research—theory, evaluation and application. To appear in *Journal of Systems and Software*.

Porter, A., Votta, L., and Basili, V. R. 1995. Comparing detection methods for software requirements inspection: A replicated experiment. *IEEE Transactions on Software Engineering* 21(6): 563–575.

Runeson, P. 2000. A new software engineering program—structure and initial experiences. *Proceedings of the 13th International Conference on Software Engineering Education & Training*. pp. 223–232.

Saaty, T. L., and Vargas, L. G. 2001. *Models, methods, concepts & applications of the analytic hierarchy process*. Netherlands: Kluwer Academic Publishers.

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

Thelin, T., Runeson, P., and Regnell, B. 2001. Usage-based reading—an experiment to guide reviewers with use cases. *Information and Software Technology* 43(15): 925–938.

Thelin, T., Runeson, P., and Wohlin, C. 2003a. An experimental comparison of usage-based and checklist-based reading. *IEEE Transactions on Software Engineering* August, 29(8): 687–704.

Thelin, T., Runeson, P., and Wohlin, C. 2003b. Prioritized use cases as a vehicle for software inspections. *IEEE Software* July/August, 20(4): 30–33.

Travassos, G., Shull, F., Fredericks, M., and Basili, V. R. 1999. 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*.

Votta, L. G. 1993. Does every inspection need a meeting? *Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering, ACM Software Engineering Notes* 18(5): 107–114.

Wohlin, C. 1997. Meeting the challenge of large-scale software engineering in a university environment. *Proceedings of the 10th Conference on Software Engineering Education & Training*. pp. 40–52.

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

Dr. Thomas Thelin is an associate professor of software engineering at the Department of Communication Systems at Lund University in Sweden. He has a PhD in Software Engineering from Lund University. His research interests include empirical methods in software engineering, software quality and verification & validation with emphasis on testing, inspections and estimation methods. He is currently working in the European project MaTeLo, which main objective is to develop an automatic generator of test cases derived from Markov usage models.

Dr. Per Runeson is an associate professor in software engineering at the Department of Communication Systems, Lund University, Sweden, and is the leader of the Software Engineering Research Group since 2001. He received a PhD from Lund University in 1998, and a MSc in Computer Science and Engineering

from the same university in 1991. He has five years of industrial experience as a consulting expert in software engineering. He was a Fulbright Research Scholar at Washington State University, WA, USA, winter 2002/03. Dr. Runeson's research interests concern methods and processes for software development, in particular methods for verification and validation, with special focus on efficient and effective methods to facilitate software quality. The research has a strong empirical focus. He has published more than 50 papers in international journals and conferences and is the coauthor of a book on experimentation in software engineering.

Dr. Claes Wohlin is a professor of software engineering at the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in Sweden. Prior to this, he has held professor chairs in software engineering at Lund University and Linkvping University. He has a PhD in Communication Systems from Lund University. His research interests include empirical methods in software engineering, software metrics, software quality and systematic improvement in software engineering. Claes Wohlin is the principal author of the book ''Experimentation in Software Engineering An Introduction'' published by Kluwer Academic Publishers in 2000. He is co-editor-in-chief of the journal of Information and Software Technology published by Elsevier. Dr. Wohlin is on the editorial boards of Empirical Software Engineering: An International Journal and Software Quality Journal. He was in October 2003 ranked among the top 15 scholars in systems and software engineering by the Journal of Systems and Software.

Thomas Olsson works as a researcher and consultant at the Fraunhofer Institute for Experimental Software Engineering in Germany. He has a Licentiate Engineering in Software Engineering from Lund University, Sweden, in 2002 and a Master of Science in Computer Science and Engineering in 1999 from the same university. His research interests lie in understanding how people work with different types of information and documentation models, especially in the context of requirements. Currently, he is involved in two European and one German funded project, being the project leader for two of them, and is at the same time pursuing a PhD degree at Lund University.

Carina Andersson is a PhD student in software engineering at the Department of Communication Systems, Lund University. She received an MSc in Engineering Physics with Industrial Economy from Lund University, Sweden, in 2001. Her main research interests include methods for effective and efficient verification and validation processes for software development.