T. Thelin, P. Runeson and C. Wohlin, "An Experimental Comparison of Usage-Based and Checklist-Based Reading", IEEE Transactions on Software Engineering, Vol. 29, No. 8, pp. 687-704, 2003.

An Experimental Comparison of Usage-Based and Checklist-Based Reading

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

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

Abstract

Software quality can be defined as the customers' perception of how a system works. Inspection is a method to monitor and control the quality throughout the development cycle. Reading techniques applied to inspections help reviewers to stay focused on the important parts of an artefact when inspecting. However, many reading techniques focus on finding as many faults as possible, regardless of their importance. Usage-based reading helps reviewers to focus on the most important parts of a software artefact from a user's point of view. This paper presents an experiment, which compares usage-based and checklist-based reading. The results show that reviewers applying usage-based reading are more efficient and effective in detecting the most critical faults from a user's point of view than reviewers using checklist-based reading. Usage-based reading may be preferable for software organizations that utilize or will start utilizing use cases in their software development.

Keywords: Controlled Experiment, Empirical Study, Reading Technique, Software Inspection.

1. Introduction

Software inspections have since its inception [8] more than 25 years ago spawned quite some interest both from the research community and industrial practice. The research includes changes to the inspection process, e.g. [4][17][25][31], support to the process, e.g. [1][7] and empirical studies, e.g. [2][34]. The suggested improvements include active design reviews [31], two-person inspection teams [4], n-fold inspections [25], phased-inspections [17], perspective-based reading (PBR) [42] and the use of capture-recapture techniques to estimate the remaining number of faults after an inspection [7]. Industry has studied the benefits of conducting software inspections [49]. Moreover, software inspections have been popularized by books on the subject [6][10].

In parallel with the development of software inspections, software engineering as such has evolved. Two directions of evolution are of particular importance in the context of this paper, namely usage-based testing [24][28][29] and the introduction of use cases in object-oriented software development [15]. One important common denominator of these two techniques that have emerged is the focus on usage. Based on this, a method for usage-based inspection was proposed Wohlin and Ohlsson and reported in a thesis [30]. The basic idea was to let the expected usage govern the inspection. The motivation behind the method is 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 is needed.

The initial idea has since been refined and further studied and some results have been presented by Thelin et al. [45][46]. These studies have refined the ideas and formulated the approach as a

new reading technique denoted *usage-based reading* (UBR). The first study was primarily focused on exploring the basic concepts of UBR as such by investigating the prioritization approach utilized in UBR [45]. The result shows that it is possible to control reviewers in order to make them focus on important parts of a software artefact. The study by Thelin et al. [46] investigate the amount of information needed to be present in the use cases in order to utilize UBR. The result shows that is beneficial to provide full scenario information to the reviewers. However, it is not shown to be worth the effort of developing all steps in the use cases solely for the purpose of UBR inspections.

The objective of this paper is to compare and hence evaluate how well the usage-based reading performs in comparison to other methods. The paper presents a controlled experiment where usage-based reading is compared to checklist-based reading (CBR). The results are promising since the study shows 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 paper is structured as follows. Section 2 gives a back ground of reading techniques for software inspections. In Section 3, the background and principles of UBR are presented. In the following sections, the experiment is presented; experiment preparation in Section 4, experiment planning in Section 5 and experiment operation in Section 6. In Section 7, the analysis of experiment data is presented and in Section 8, the results are discussed. Finally, a summary is presented in Section 9.

2. Reading Techniques

In order to improve the comprehension and the fault searching process of software inspections, different reading techniques can be utilized. Examples of suggested reading techniques are *checklist-based* [8], *perspective-based* [42], *defect-based* [34], *usage-based* [45] and *traceability-based* [47] reading. The reading techniques have been developed based on different assumptions regarding inspections. However, they have a common general goal, which is to help reviewers to become focused when inspecting a software document and thereby to detect more faults. A summary of the reading techniques and their specific purpose is provided in this section.

When no specific reading technique is used, it is denoted *ad hoc*, i.e. the reviewers do not get any support of how to find faults. Hence, reviewers detect faults based only on their personal skill and experience.

2.1 Checklist-Based Reading

Checklist-based reading (CBR) is a reading technique, where the reviewers applying CBR use a list of issues helping them to know what kinds of faults to look for. The checklist items can either be expressed in ordinary statements or as questions.

An example of checklist items is provided in Table 1, which contains a subset of the checklist used for an experiment comparing perspective-based and checklist-based reading [21]. The checklist was used to detect faults of the types consistency, correctness and completeness.

TABLE 1. An example of checklist items, which was used by Laitenberger et al. [21] in an experiment checking a design specification. The original checklist consists of 19 items.

No.	Where to look?	How to detect?
1.	All Design Diagrams	Is each name unique?
2.		Are all names used in the diagram consistent?

TABLE 1. An example of checklist items, which was used by Laitenberger et al. [21] in an experiment checking a design specification. The original checklist consists of 19 items.

No.	Where to look?	How to detect?
3.		Are all names used in the diagrams correct?

Today, CBR is considered to be the standard reading technique in software organizations [20]. Therefore, CBR is often used as a baseline method in empirical studies when investigating reading techniques, see Table 2 and Table 3.

2.2 Defect-Based Reading

Defect-based reading (DBR) is a reading technique that is focused on detecting specific types of faults [34]. DBR defines a set of scenarios, which are procedures that a reviewer follows during inspection. DBR is aimed at detecting the same types of faults as CBR. However, more information is included in the scenarios in order to design a more structured reading technique. The technique was designed by Porter et al. in order to help reviewers to detect faults in requirements documents specified in the *software cost reduction* (SCR) tabular requirements notation [33]. The initial experiment has since then been replicated by other researchers, see Table 2.

TABLE 2. Summary of studies investigating defect-based reading.

Study	Purpose	Environment	Subjects	Significant?
Porter et al. [34] – 1995	DBR vs. Ad hoc and CBR	Academic	24+24	YES
Fusaro et el. [9] – 1997	DBR vs. Ad hoc and CBR	Academic	30	NO
Miller et al. [26] – 1998	DBR vs. CBR	Academic	50	Inconclusive
Sandahl et al. [40] – 1998	DBR vs. CBR	Academic	24	NO
Porter et al. [35] – 1998	DBR vs. Ad hoc and CBR	Industrial	18	YES

The scenarios used in these studies were derived by Porter et al. [33] from a checklist that was used for inspecting requirements documents. Three different types of scenarios were used, representing classes of faults. The scenarios were aimed at detecting *data type inconsistencies*, *incorrect functionality* and *ambiguities or missing functionality.* Hence, each scenario is a subset of the checklist, but includes more information of how to locate faults.

Each scenario is designed to be more specific than the checklist, which in its turn should be more structured than ad hoc reading. Since the scenarios should be more specific, they do not cover all faults in the documents, which means that several (in this case three) perspectives are needed to cover all faults (see Figure 1). In the initial experiment by Porter et al., the authors estimated that 50% of the faults were covered by each scenario.

As seen in Table 2, the results of the different studies do not point in the same direction. Two out of five experiments show that DBR is more effective than CBR, and two of them do not show significant result. The study by Miller et al. [26] is inconclusive in the sense that the result is significant for one of the documents inspected, but not for the other one. These studies investigate effectiveness from an individual and a team perspective, which means that the detection rates are investigated (a few of them also consider the time variable).

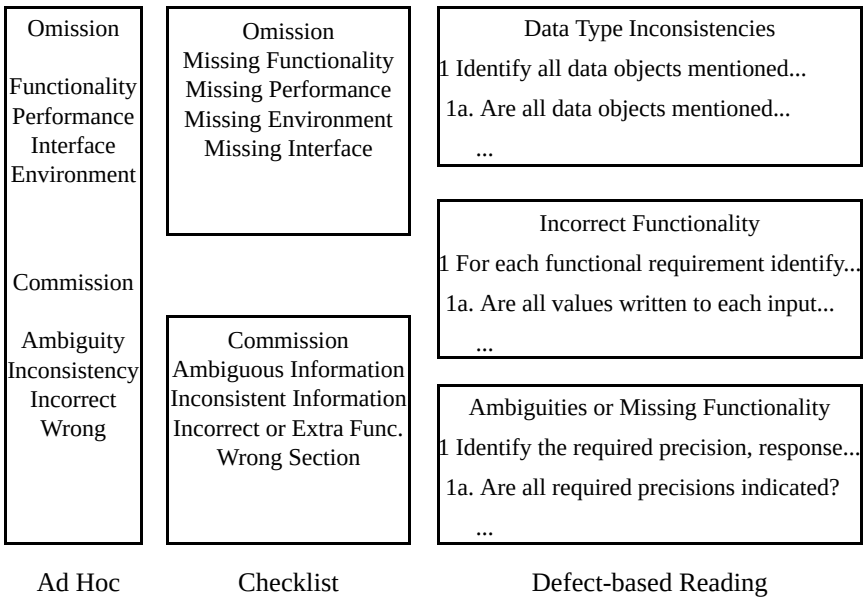

FIGURE 1. The relationship between the reading techniques, ad hoc, checklist-based and defect-based reading. The vertical extent represents the coverage and the horizontal extent represents the degree of detail [34].

An attempt to systematically address the combined knowledge, gained from experiments and replications is reported by Hayes [12], where *meta-analysis* is applied to the results of the studies in Table 2. It is concluded from the meta-analysis that the effect sizes for the inspection methods are inhomogeneous across the experiments. The studies by Porter et al. (1995 and 1998) show most similar results, and an interpretation of the meta-analysis identifies characteristics which make them different from the other three studies: (1) they are conducted in a context where the subjects are more familiar with the notation used, (2) they are conducted in the US where cruise control (one of the requirements documents inspected was a cruise control document) are more common in cars than in Europe where the other three studies are performed. These hypotheses are, however, not possible to test with the given data, and thus more experimentation is needed.

2.3 Perspective-Based Reading

Perspective-based reading (PBR) assigns different perspectives to the reviewers to apply when inspecting a software artefact. The basic assumptions of PBR are (1) that a reviewer with a specific focus performs better than a reviewer with the responsibility to detect all types of faults, and (2) that different foci can be designed so that their union yields full coverage of the inspected artefact. Another important assumption for PBR is that it is easier to detect faults if a reviewer works in a structured manner and reads actively. Further, by utilizing the artefact under inspection in a way that it will be used in subsequent phases, the reviewer is able to detect faults that otherwise are not detected until later. Basili et al. [1] use designer, tester and user as inspections roles. Note that the above assumptions are not only true for PBR. For example, assumption one is also stated for DBR and UBR, and number two is one of the basics of DBR.

Model building is a central part of the three perspectives used for PBR. The models used for inspecting a document stem from well-known techniques used in the different software phases. Designers utilize structured analysis, testers utilize equivalence partitioning and users utilize use cases. All these models are structured and adapted to be used as perspectives. By combining the three perspectives, the resulting inspections are expected to achieve better fault coverage of the

inspected document and less overlap among what different reviewers find. Hence the reviewers are expected to be more cost-effective.

PBR has been extensively empirically evaluated for requirements documents, see Table 3. Besides requirements documents, PBR has been adapted and empirically evaluated for design inspections [19][21], code inspections [18] and usability inspections [52].

TABLE 3. Summary of studies investigating perspective-based reading for requirements documents.

Study	Purpose	Environment	Subjects	Significant?
Basili et al. [1] - 1996	PBR vs. Ad hoc	Industrial	12+13	YES
Ciolkowski et al. [5] - 1997	PBR vs. Ad hoc	Academic	25+26	YES
Sørumgård [44] - 1997	PBR vs. PBR2	Academic	48	NO
Shull [41] - 1998	PBR vs. Ad hoc	Academic	66	YES
Regnell et al. [36] - 2000	Are different faults detected by different perspectives in PBR?	Academic	30	NO
Lanubile and Visaggio [22] - 2000	PBR vs. Ad hoc and CBR	Academic	114+109	NO
Biffl [3] - 2001	PBR vs. CBR	Academic	169	NO
Halling [11] - 2001	PBR vs. CBR	Academic	177	NO

2.4 Traceability-Based Reading

Traceability-based reading (TBR) [47] is a reading technique adapted for inspecting object-oriented design specifications. The method is divided into vertical and horizontal reading, The purpose of the vertical reading is to check whether the design corresponds to the requirements, whereas the horizontal reading checks the different design artefacts (e.g. class diagrams, package diagrams and state machine diagrams) against each other.

3. Usage-Based Reading

Many reading techniques focus on finding as many faults as possible, regardless of their importance. The inspection effectiveness is often measured in terms of number of faults found, without taking into account that some faults in the inspected object are likely to affect the final system quality more than others do.

The principal idea behind usage-based reading (UBR) is to focus the reading effort on detecting the most critical faults in the inspected object. Hence, faults are *not* assumed to be of equal importance, and the UBR method is aimed at finding the faults that have the most negative impact on the users' perception of system quality. The UBR method focuses the reading effort guided by a prioritized, requirements-level use case model during the individual preparation of software inspections. Hence, existing procedures for resource scheduling, meetings and follow-up can be used in conjunction with UBR

In order to specify the users' perception of system quality, use cases are prioritized. The order of the use cases reflects what a user or a group of users thinks is most important in the system to be developed. The prioritization can be made by, for example, pair-wise comparisons according to the *analytic hierarchy process* [39]. The use cases are prioritized before an inspection session and it should be made by some potential users or by someone who is familiar with the usage of

the software. The use cases can be utilized for all inspections (requirements, design, code, etc.) in a specific project. Hence, the prioritization only has to be done once for a software project. The reviewers actively read the document under inspection by manually executing the use cases and try to detect faults which are most important according to the prioritization, and hence to the users.

The background of UBR is from operational profile testing [29] and the user perspective in object-oriented development [15]. UBR utilizes a set of use cases as a vehicle for focusing the inspection effort, much the same way as a set of test cases focuses the testing effort. The use cases tell the reviewers how to inspect a design or code document in a similar manner as the test cases tell the testers how to test the system. The individual inspection of a design document using UBR is performed in the following basic steps:

- Before inspection – Prioritize the use cases in order of importance from a user's point of view.
- Preparation – Glance through the design document to be inspected, the use cases utilized to guide the reading and the requirements document. The requirements document is used as a reference to which the design is verified.
- Individual inspection – Inspect the design document by following the procedure:
  (1) Select the use case with the highest priority.
  (2) Trace and manually executing the use case through the design document and use the requirements document as a reference.
  (3) Ensure that the document under inspection fulfils the goal of the use case, that the needed functionality is provided, that the interfaces are correct etc. Identify and report the issues found.
  (4) Repeat the inspection procedure using the next use case, until all use cases are covered, or until a time limit is reached.

Two variants of the UBR method are defined, *ranked-based reading* and *time-controlled reading.* The former prioritizes the use cases with respect to the importance from a user's perspective. A reviewer using ranked-based reading follows the use cases in the order they appear in the ranked use case document. Time-controlled reading adds a time budget to each use case in order to force a reviewer to utilize a specific use case the specified time. Time budgets are applied to each use case and are normally longer for use cases given a high rank and less for use cases given a lower rank.

UBR is a novel reading technique, which differs from the other reading techniques. Although UBR is related to PBR, several differences exist. The relation to PBR is the utilization of the user perspective. However, UBR focuses only on the users and guides the reviewers based on the users' needs during an inspection by providing the reviewers with developed and prioritized use cases. In PBR, different perspectives are used to produce artefacts during an inspection. The reviewers applying the user perspective develop use cases based on the inspected artefact and thereby find faults. In UBR, the use cases are used as a guide through the inspected artefact. The differences are hence that reviewers applying UBR *utilize* existing and prioritized use cases while reviewers applying PBR actively *develop* non-existing use cases. The goal of UBR is to improve efficiency and effectiveness by directing the inspection effort to the most important use cases from a user's viewpoint, while PBR has the goal of improving effectiveness by minimising the overlap among the faults that the reviewers find. The latter is, however, not always achieved [36].

Another practical difference exists between PBR and UBR. PBR is a reading technique that is applicable to all artefacts, i.e. the scenarios developed for PBR are general. (In PBR, the term scenario is a meta-level concept, denoting a procedure that a reader of a document should follow during an inspection.) Thus, scenarios developed for requirements documents may be used for all requirements documents. However, the same scenarios cannot be used for design or code inspec-

tions. On the contrary, UBR scenarios are specific to each project, which means that the use cases can only be utilized within the project they are developed for. However, they can be used for requirements, design as well as for code inspections in that project. In addition, they may also be used for test specification development and inspection.

In the investigation in this paper, ranked-based reading is used. To investigate the effects of using UBR, the following research questions are addressed:

- 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?
- RQ4 – Is UBR more effective and efficient than CBR considering the performance of an inspection team?

4. Experiment Preparation

This section describes the preparation needed to conduct the experiment and the subjects acting in the experiment. The experiment is based on an experimental package developed at Lund University [45].

4.1 Reviewers

The students participating as reviewers in the study were 23 fourth-year Software Engineering Master's students at Blekinge Institute of Technology in Sweden. Many of the students have extensive experience from software development. As part of their bachelor degree, they have obtained extensive practical training in software development. They have, for example, participated in a one semester project comprising 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's 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.

The experiment was a mandatory part of a course in verification and validation. The course included lectures and assignments both related to verification and validation of software products and evaluation of software processes. The latter means that the students have been introduced to empirical studies and the opportunity of using them to evaluate different techniques and methods. The objective of the experiment, from an educational perspective, was that the students should be exposed to an empirical study in software verification and validation at the same time as they were introduced to some of the on-going research in the area.

4.2 Inspection Material

The inspection experiment is based on material developed for a verification and validation course in software engineering at Lund University in Sweden. The material consists of four documents in structured text: one requirements document, one design document written in the *specification and description language* (SDL) [13] (9 pages, 2300 words), one use case document, and one checklist. The use case document and the design document were used in a previous experiment at Lund University. The requirements document and the checklist were developed for this experiment.

The requirements document was written in natural language (English). The document is used as a reference document to show how the system is meant to work. The checklist consists of 18 check

items (provided in appendix) and is based on a checklist presented by Laitenberger et al. [21]. It would have been preferable to use a checklist from an industrial application to check this kind of design, but no such checklist was found. Therefore, we used a modified version of a checklist utilized in experiments with the purpose of comparing CBR and PBR. The checklist items were modified to fit the taxi management system and they were sorted in order of importance.

The subjects inspected the design document using the requirements document as a reference. To guide the reading they used either a use case document or a checklist. The design consists of software modules of a taxi management system and descriptions of signals in-between these modules. The modules are one taxi module for each vehicle, one central module for the operator and one communication link, see Figure 2. Furthermore, the design document consists of two *message sequence charts* (MSC) [14], which show signalling among the modules for two different cases, one normal case and one special case. The use cases are written in *task notation* [23] (see Figure 3) and are prioritized using the *analytic hierarchy process* (AHP) [39] from a user's point of view, i.e. the function of the first use case is the most important to the user while the last use case is least important. The use case document contains 24 use cases.

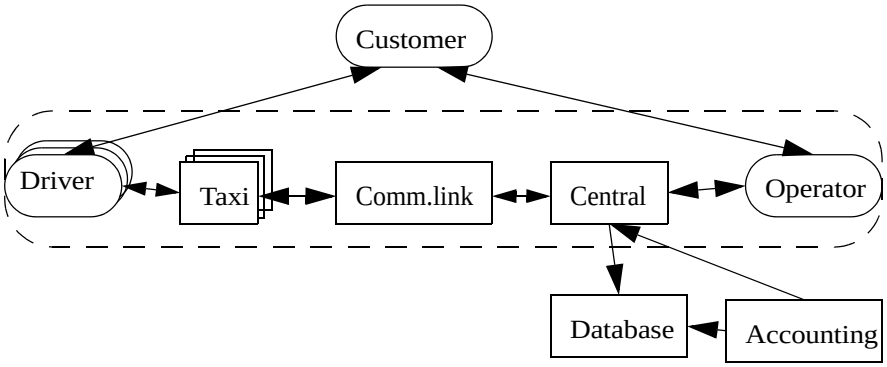

FIGURE 2. The taxi management system. The boxes represent software modules and the circles represent users. The software for the database and accounting system is not implemented in the first version.

Taxi: Log in

Purpose: The driver logs on to the system to acknowledge presence and to be able to receive orders.

Tasks:
1. The car is in state "Offline".
2. The driver swipes the identification card in the terminal in the car.
3. The terminal sends the position and driver information to the central.
4. The central confirms the log in.
5. The car sends the position (zone).
6. The central starts sending overview information on all zones.
7. The car is in the state "Available".

Variants:
1b. The car is not in the state offline. There can be only one driver logged in at the same time in the car terminal. The car has to be in state "Offline" in order to allow new logins.
4b. The card is not valid. The central rejects the driver. The driver and the car are not logged in and remain in state "Offline".

FIGURE 3. Example of a use case written in task notation [23]. The use case describes the login procedure for a taxi driver.

The design document contains 38 faults, of which two are new faults found during the experiment and eight are seeded faults injected by the person who developed the system. The 28 others

are faults made during development of the design document and later found in inspection or test. These faults were re-inserted prior to the experiment.

4.3 Roles and Activities

The development of the documents and the design of the experiments have involved six 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 experiment.

The activities during the experiment planning are described below and summarized in Table 4.

- Development – One person was responsible for the development of the taxi management system including requirements specification, design, code and user documentation. A second person developed the textual requirements. A third person developed the checklist. All developed artefacts were inspected in order to achieve as high quality as possible. The faults were logged during development and then seeded back into the design document before the experiment.
- Prioritization of use cases – Three persons prioritized the use cases independently of each other. The prioritization was made by pair-wise comparison of each use case using the analytic hierarchy process (AHP) [39]. Then a fourth person combined the priorities and ordered the prioritized use cases. The result from AHP is a ranking of the use cases. The three persons who made the ranking did not fully agree on the order of the use cases. However, the disagreement was small. The fourth person calculated a mean of the others' ranking to achieve a final order of the use cases. The three persons who performed the prioritization are experienced in the taxi management system in the sense that they have taken part in the requirements elicitation together with a real customer. In addition, two of them have given a course where an extended requirements document for a similar system was developed. The instruction to them was to pair-wise compare the use cases under the criterion which are most important for a user of the system.
- Classification of faults – Two persons classified the faults according to the same criterion as for the use cases using AHP, i.e. from a user's point of view. Afterwards, a third person checked and confirmed the fault classification.
- Design and analysis of the experiment – One person was responsible for the design of the experiment and the analysis of the data. The fault classifications as well as the data analysis were validated by two other persons.

TABLE 4. Summary of roles and activities during the preparation and execution of the experiment.

Role	Activities
Developer 1	Developed the system (requirements, design, code and user description)
	Logged faults during development
	Seeded eight faults
Developer 2	Developed the textual requirements specification
Analyser	Designed experiment
	Conducted experiment
	Analysed data
	Classified faults
	Developed checklist

TABLE 4. Summary of roles and activities during the preparation and execution of the experiment.

Role	Activities
Taxi Expert 1	Prioritized use cases
	Developed checklist
	Checked classification of faults
Taxi Expert 2	Prioritized use cases
	Classified faults
Taxi Expert 3	Prioritized use cases

4.4 Fault Classification

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

- Class A faults – The functions affected by these faults are crucial for the user, i.e. the functions affected are important for the 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 the 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 functions affected by these faults are *not* important for the user. An example of this kind of fault is: a signal is missing in a table summarizing all signals, but it is correctly defined in the section that describes the signals.

The design document contains 13 *class A faults*, 14 *class B faults* and *11 class C faults*. No syntax errors like spelling errors or grammatical errors were logged as faults. If these kinds of errors were found, they were not included in the analysis. Three persons made the classification of the faults prior to the experiment.

5. Experimental Planning

5.1 Variables

Three types of variables are defined for the experiment, *independent*, *controlled* and *dependent* variables.

- Independent Variable – The independent variable is the reading technique used. The experiment groups used either UBR or CBR.
- 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 comprising seven questions.
- Dependent Variables – The dependent variables measured are time and faults. The four first variables are direct measures. The three last are indirect measures and are calculated using the direct measures.

1. Time spent on preparation, measured in minutes.
2. Time spent on inspection, measured in minutes.
3. Clock time when each fault was found, measured in minutes, measured from start of preparation.
4. Number of faults found by each reviewer.

5. Number of faults found by each experiment group.
6. Efficiency (faults/hour), measured as: $60 \cdot \frac{\text{Number of Faults Found}}{\text{Preparation Time + Inspection Time}}$.
7. Effectiveness (detection rate), measured as: $\frac{\text{Number of Faults Found}}{\text{Total Number of Faults}}$.

5.2 Hypotheses

The general hypothesis of the experiment is that UBR is more efficient and effective in finding faults of the most critical fault classes, i.e. UBR is assumed to find more faults per time unit, and to find a larger rate of the critical faults.

The dependent variables are analysed to evaluate the hypotheses of the experiment. The main null and alternative hypotheses [27] are stated below. These are evaluated for all faults, class A faults and class A&B faults. The hypotheses concern efficiency, effectiveness and fault detecting differences:

- $H_{0\ Eff}$ – There is no difference in *efficiency* (i.e. found faults per hour) between the reviewers applying prioritized use cases and the reviewers using a checklist.
- $H_{A\ Eff}$ – There is a difference in *efficiency* between the reviewers applying prioritized use cases and the reviewers using a checklist.
- $H_{0\ Rate}$ – There is no difference in *effectiveness* (i.e. rate of faults found) between the reviewers applying prioritized use cases and the reviewers using a checklist.
- $H_{A\ Rate}$ – There is a difference in *effectiveness* between the reviewers applying prioritized use cases and the reviewers using a checklist.
- $H_{0\ Fault}$ – The reviewers applying prioritized use cases do not *detect different* faults than the reviewers using a checklist.
- $H_{A\ Fault}$ – The reviewers applying prioritized use cases *detect different* faults than the reviewers using a checklist.

The hypotheses are summarized in Table 5.

TABLE 5. Summary of hypotheses investigated in this study.

Hypothesis / Fault class	All (A+B+C)	A	A+B
Efficiency	$H_{Eff,\ all}$	$H_{Eff,\ a}$	$H_{Eff,\ b}$
Effectiveness (Rate)	$H_{Rate,\ all}$	$H_{Rate,\ a}$	$H_{Rate,\ b}$
Different (Fault)	H_{Fault}	–	–

5.3 Design

The students were divided into two groups, one group using UBR and one group using CBR. Using the controlled variable to get a block design, the students were divided into three groups and then randomized within each group, resulting in 11 students in the UBR group and 12 students in the CBR group. A questionnaire with seven questions was used to explore the students' experiences in programming, inspections, SDL, use cases and taxi systems. The questionnaire showed that the students had three types of backgrounds based on the experience. Therefore, it was obvious to divide them into these groups and thereby block the experience factor of the experiment.

The instrumentation of the experiment consists of one requirements document, one design document, one use case document, one checklist and one inspection record. The inspection record contains fields to collect all the data used to analyse the experiment.

The experiment data are analysed with descriptive analysis and statistical tests [16][51]. The collected data were checked whether it follows a normal distribution. Since no such distribution could be demonstrated using normal probability plots and residual analysis, nonparametric tests are used. The Mann-Whitney test [43] is used to investigate hypotheses H_{Eff} and H_{Rate} and a chi-square test is used to test H_{Fault}.

5.4 Threats to Validity

The threats to the validity of the experiment are analyzed below. As the purpose of the study is to compare two reading techniques, and more studies are needed for generalization purposes, the threats to internal and construct validity are most critical. When trying to generalize the results to a more general domain, the external validity becomes more important [51].

Threats to *conclusion validity* are considered under control. Robust statistical techniques are used, measures and treatment implementation are considered reliable. The only risk in the treatment implementation is that the subjects were trained one day and the experiment was conducted the next day. Hence, they might inform each other about the other technique, even though they were strictly forbidden to do so. However, nobody would gain from doing so and hence the risk of doing it is low. Random variation in the subject group is blocked, based on the controlled variable.

Concerning the *internal validity*, the risk of rivalry between groups is considered the largest one. However, the student subjects were informed that they would be given additional training in the other reading technique used on the second day (see schedule in Section 6). Further, their grade on the course was not affected by the performance in the experiment, only on their attendance. This could lead to lack of motivation. However, the students attended a verification and validation course, where one topic was empirical research. The students were introduced and motivated to empirical research before the experiment and a debriefing session was held after the analysis was ready. In order to discuss the results in the course, they had to give good inputs to the empirical work. Furthermore, the students were randomly assigned. Thus, the threat of lack of motivation was minimized in the experiment.

Threats to the *construct validity* are not considered very harmful. The development of the textual requirements document was performed after the development of the use cases. Hence, there is a risk that the use cases may have affected the requirements document to make it suitable for the use cases. On the other hand, the inspection object was the design document and the requirements document was just a reference.

Concerning the *external validity*, the use of students as subjects is a threat. However, the students are fourth year master students in software engineering, and a large share of the students have part time jobs in software companies, hence being more representative of software industry than students in general. Another threat to the external validity is the design document used in the experiment. Only one domain is considered and the size of the inspected document is in the smaller range for real-world problem, even though it describes a real-world problem.

6. Experimental Operation

The experiment was run over two days during spring 2001, see schedule in Table 6. During the first day, the students were given an introductory presentation of the taxi management system. All students were listening to the same presentation of the system. This presentation included all

material that was the same for both groups. Then they were divided into two groups depending on the method they were going to use during the inspection experiment. The second session included a brief presentation of the reading technique (either CBR or UBR) 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.

During three hours of the second day, the students conducted the inspection experiment. After this session, a brief follow-up meeting was held, where the UBR group tried CBR and vice versa. The main purpose of this session was educational, but also served as a follow-up discussion forum for the inspection experiment.

The package for the experiment contained:

1. Inspection record, including:
   - a description of the fault classification scheme (class and severity)
   - a time log for preparation and inspection time
   - a fault log. For each fault found, the reviewers logged the use case / checklist item used, the clock time when found, the class and severity of the fault
2. a requirements document
3. a design document
4. either a use case document or an inspection checklist.

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 everything is checked or after 2 hours and 45 minutes. When finished, verify that the logged data seem to be correct and hand them in.

After each student handed in their inspection record, an inspection moderator (one of the authors) checked for errors and missing data in the record in order to get as accurate data as possible.

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.

TABLE 6. Schedule for the Experiment.

	CBR group	UBR group
Day 1 (1.15 p.m. - 2.00 p.m.)	General introduction to the Taxi Management System	
Day 1 (2.15 p.m. - 3.00 p.m.)	Introduction to CBR	Introduction to UBR
Day 2 (9.15 a.m. - 12.00 p.m.)	Inspection Experiment	
Day 2 (1.15 p.m. - 2.00 p.m.)	Introduction to UBR and follow-up discussion	Introduction to CBR and follow-up discussion

7. Analysis

This section presents the data collected during the experiment and pinpoints important issues to discuss in Section 8, where the results are discussed. First, a descriptive analysis is carried out and then the statistical analyses are presented.

7.1 Time Data

During the experiment, each reviewer logged the clock time of each activity performed. They logged start time, end time, reading time, inspection time, time when they found a fault and break times. Since the experiment was limited in time, all reviewers spent almost the same amount of time. The only differences among the reviewers are the break times and the proportion of time spent reading through the documents related to the time spent on pure inspection.

The differences between the groups are small; none of the differences are statistically significant. In Table 7, the mean and standard deviation of the time spent by the reviewers are presented. The reviewers in the checklist group spent a little more time both for reading and inspection. In total, they spent on average 10 minutes more than the reviewers in the UBR group. This means that an average reviewer had the opportunity to find about one more fault, since it took about 11 minutes to find one fault. However, they spent most of the additional time on preparation.

TABLE 7. Mean and standard deviation values for preparation and inspection time (minutes).

	Mean		Standard Deviation	
	UBR	CBR	UBR	CBR
Preparation	52.8	59.3	20.4	15.5
Inspection	77.1	81.1	17.8	19.2
Total	129.9	140.4	14.5	12.4

7.2 Time versus Faults

When the reviewers found a fault, they logged the clock time in the inspection record. In Figure 4, the cumulative fault detection is shown. The plot is standardized with respect to the number of reviewers in each group, i.e. it represents an average reviewer. The mean preparation time for the UBR group and the CBR group was 53 and 59 minutes respectively.

The reviewers in the UBR group started to find faults earlier than reviewers in the CBR group. The difference is about 20 minutes. Reviewers in the UBR group started to find faults after 54 minutes and the CBR group started to find faults after 74 minutes. (Note that these are mean values when one fault has been found). This difference could either be due to it being easier to start

with use cases than with a checklist item, or due to reviewers in the UBR group spending shorter time reading through the documents.

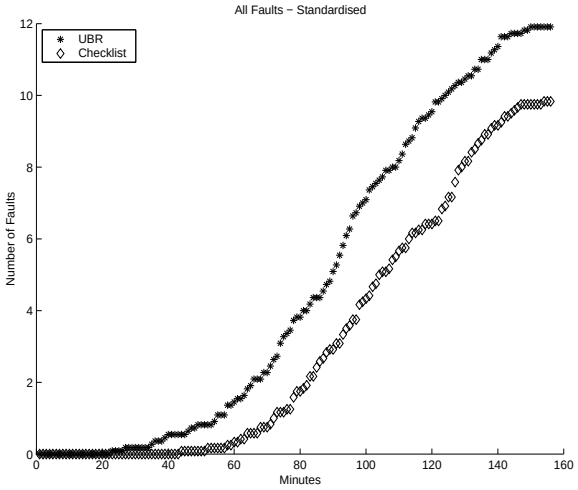

FIGURE 4. The cumulative number of faults found during the inspection. The data are standardized by the number of reviewers in each group.

In Figure 5 and Figure 6, class A faults and class A&B faults are plotted versus detection time and standardized with the number of reviewers. It took a little longer time before the identification of class A faults started and the difference between the UBR and CBR with respect to when the first class A fault found is more than 20 minutes. The increase is not as evident if both A&B-faults are considered.

In both figures, the slope of the UBR curve increases faster than the curve for the CBR group. This indicates that more severe faults are found more efficiently as well as effectively by reviewers in the UBR group.

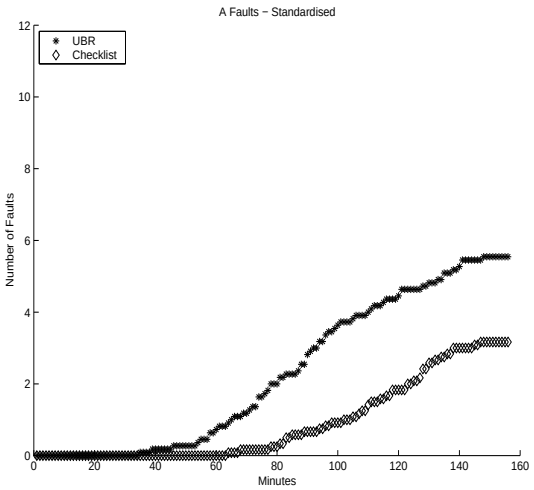

FIGURE 5. The cumulative number of class A faults found during the inspection. The data are standardized by the number of reviewers in each group.

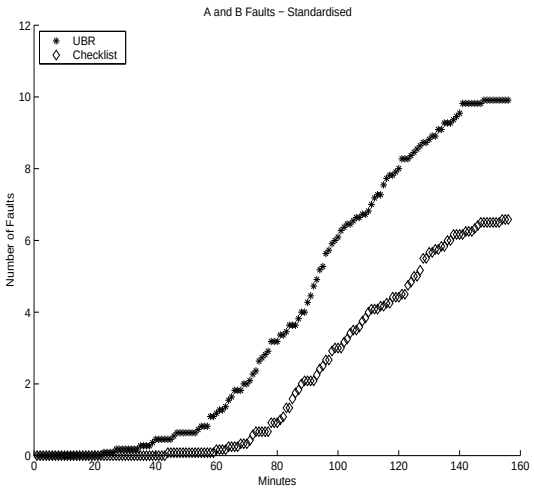

FIGURE 6. The cumulative number of class A&B faults found during the inspection. The data are standardized by the number of reviewers in each group.

In Table 8 and Table 9, the efficiency and effectiveness of an average reviewer for UBR and CBR are presented. An average reviewer is calculated as the sum of the efficiency (effectiveness) of the reviewers for each group, and then standardized with the size of the group. A UBR reviewer is more efficient and effective than a CBR reviewer for all classes of faults except for class C faults. Reviewers using UBR found about twice as many class A faults than a CBR reviewer per hour and 70% more class A&B faults per hour. A reviewer applying UBR found 75% more class A faults and 51% more class A&B faults than a CBR reviewer. A similar pattern is shown for class B faults. However, for class C faults, CBR reviewers were more efficient as well as more effective. (The differences between the efficiency (effectiveness) figures and the percentage values are due to non-rounded figures are used in the percentage calculation).

TABLE 8. A comparison of the efficiency for an average reviewer.

	Mean		Standard deviation		More faults found / hour
	UBR	CBR	UBR	CBR	
All Faults	5.6	4.1	2.0	2.0	35% (UBR)
Class A Faults	2.6	1.3	1.0	1.1	95% (UBR)
Class B Faults	2.1	1.4	1.2	0.7	46% (UBR)
Class C Faults	0.9	1.4	0.4	0.8	49% (CBR)
Class A&B Faults	4.7	2.8	1.8	1.7	70% (UBR)

TABLE 9. A comparison of the effectiveness for an average reviewer. The figures in parentheses are the total number of faults in the classes.

	Mean		Standard deviation		More faults found
	UBR	CBR	UBR	CBR	
All Faults (38)	0.31	0.25	0.09	0.14	21% (UBR)
Class A Faults (13)	0.43	0.24	0.17	0.21	75% (UBR)
Class B Faults (11)	0.31	0.24	0.15	0.13	28% (UBR)
Class C Faults (14)	0.18	0.30	0.08	0.21	63% (CBR)
Class A&B Faults (27)	0.37	0.24	0.12	0.16	51% (UBR)

7.3 Fault Data

Another view of the data is to investigate which reviewers found the specific faults. The intention behind UBR is to guide the reviewers to find the most critical faults with respect to usage. By analysing which faults are found with the methods respectively, it can be investigated whether the two methods reveal the same faults or different sets of faults.

In Figure 7, class A faults are show, in Figure 8 class B faults and in Figure 9 class C faults are shown. The actual number of reviewers that found each fault is shown on the y-axis. All class A faults are found by at least one of the reviewers in the UBR group. 85% of these faults are found by at least one CBR reviewer. However, more reviewers find the class A faults if UBR is used as a reading technique, i.e. the detection probability is higher. The same conclusion can be drawn for class B faults. However, for C faults it is more difficult to find a pattern and draw conclusions about the different techniques.

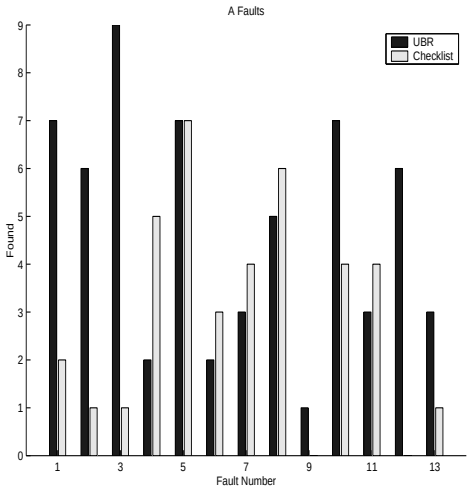

FIGURE 7. The distribution of the number of reviewers that found each class A fault.

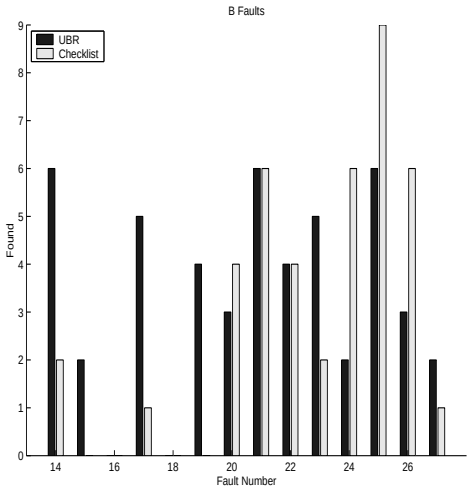

FIGURE 8. The distribution of the number of reviewers that found each class B fault.

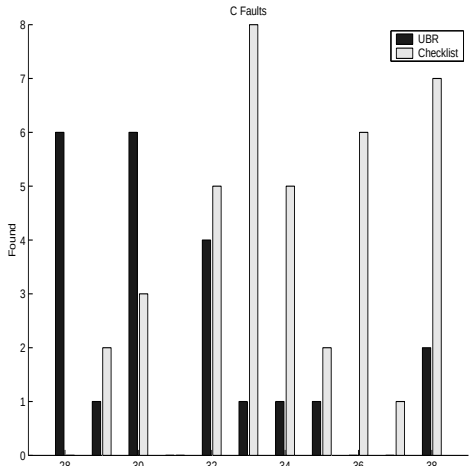

FIGURE 9. The distribution of the number of reviewers that found each class C fault.

Table 10 reports which technique found most unique faults, and the percentage figures show how much more is found compared to the other technique. For all faults, class A and class B faults, the UBR group found more unique faults. They found 18% more unique class A faults and 20% more class B faults. However, CBR found 13% more unique C faults. In total, UBR missed 13% of the faults and CBR missed 21% of the faults (all faults considered).

TABLE 10. A comparison of the number of unique faults found.

	More Unique Faults
All Faults	10.0% (UBR)
Class A Faults	18.2% (UBR)
Class B Faults	20.0% (UBR)
Class C Faults	12.5% (CBR)
Class A&B Faults	19.0% (UBR)

7.4 Effectiveness and Efficiency

The most important characteristic of a reading technique is whether it is efficient and effective enough. Efficiency is defined as the number of faults found per hour and effectiveness is defined as the fraction of the number of faults found per total number of faults in the inspected document. This section provides box plots of these parameters together with statistical analysis of the performance.

In Figure 10 and Figure 11, box plots of the efficiency and the effectiveness are shown. UBR is more efficient as well as more effective than CBR. These figures show the same facts as discussed earlier in this section. This is true for all faults, class A faults and A&B faults.

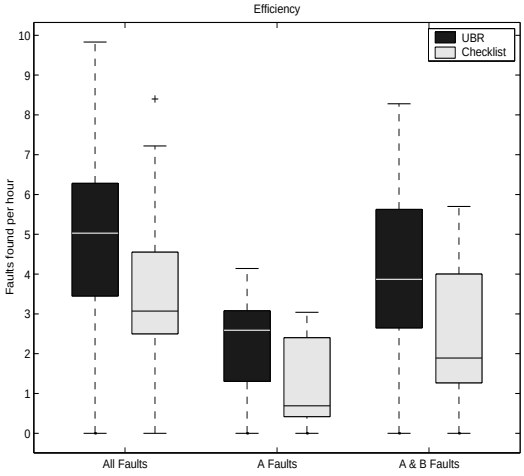

FIGURE 10. The efficiency for all faults, class A faults and class A&B faults.

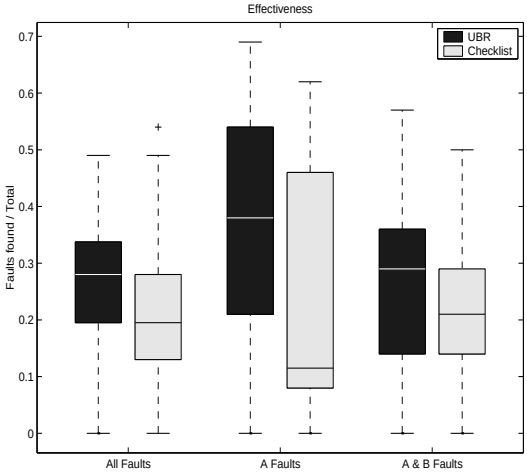

FIGURE 11. The effectiveness for all faults, class A faults and class A&B faults.

In Table 11, p-values for the Mann-Whitney test are shown. The UBR group is significantly more efficient than the CBR group for class A, class B, class A&B faults. The group is also significantly more effective for class A and class A&B faults. For the rest of the classes, no significant differences can be demonstrated. Hence, the statistical analysis combined with the descriptive analysis show that using UBR is significantly more efficient and effective with respect to critical faults from a user's perspective.

TABLE 11. P-values for Mann Whitney tests of Efficiency and Effectiveness (α=0.05). Statistical significance is marked with an (S), non-significant (–).

	Efficiency (p-value)	Effectiveness (p-value)
All Faults	0.042 (S)	0.103 (–)
Class A Faults	0.013 (S)	0.036 (S)
Class A & B Faults	0.016 (S)	0.031 (S)
Class B Faults	0.148 (–)	0.175 (–)
Class C Faults	0.268 (–)	0.148 (–)

To test whether the two groups found different faults (H_{Fault}), a chi-square test is used [43][36]. The p-value is equal to 0.001, which means that the two groups find different faults.

7.5 Team Performance

Although individuals perform inspections, the combined result of an inspection team is the important outcome of an inspection session. Since the reviewers may find the same faults, they may not add as much to the team performance [32]. A simulation of the inspection meeting is

performed (nominal teams) to investigate the meeting performance of the reading techniques. The purpose of the simulation is to investigate whether a UBR team, a CBR team or a mixed team is the best alternative when performing inspections. The purpose is not to find the ultimate team size, but to analyse the composition of a team. In order to find the best team, trade-off analysis between efficiency and effectiveness has to be done, which is out of scope of this investigation.

When real inspection meetings are performed, new faults might be detected during the meeting (defined as meeting gain). Meeting loss may also occur, which is defined as the number of faults that are considered not to be faults in the meeting, although they are. Since this is a simulation of inspection meetings, i.e. only nominal teams are investigated, no such data as meeting gain or loss can be calculated. Hence, the box plots can only be used as a comparative study. This means that the effectiveness and efficiency values cannot be treated as absolute values without the assumption that no gains or losses are involved during the inspection meeting.

To investigate the team performance, all possible combinations were made and the result is shown in Figure 12 and Figure 13. The box plots show combinations of reviewers only in the UBR group, only in the CBR group and a combination of reviewers in both groups. For example, for the two-inspection-teams, one reviewer from each group is used in the mixed teams. Since we have shown that the groups detect different faults, these mixed teams could give better results.

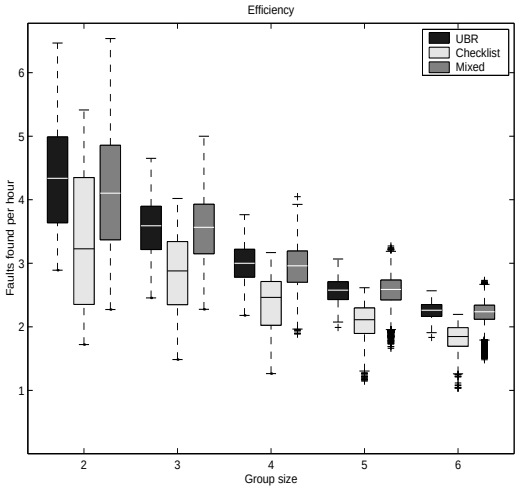

FIGURE 12. Efficiency for different group sizes. All faults are included.

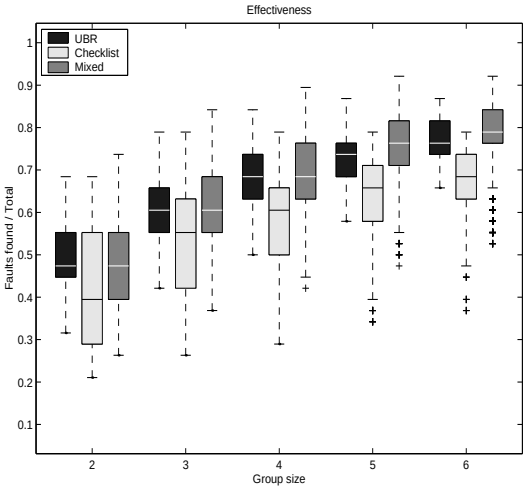

FIGURE 13. Effectiveness for different group sizes. All faults are included.

For all team sizes, UBR teams perform better than CBR teams. UBR teams outperform the mixed teams in all cases except for effectiveness in team sizes 5 and 6. However, there are only small differences.

Similar results are obtained when class A faults and class A&B faults are observed in Figure 14 to Figure 17. However, in these cases, the UBR team is better than the mixed team for all team sizes.

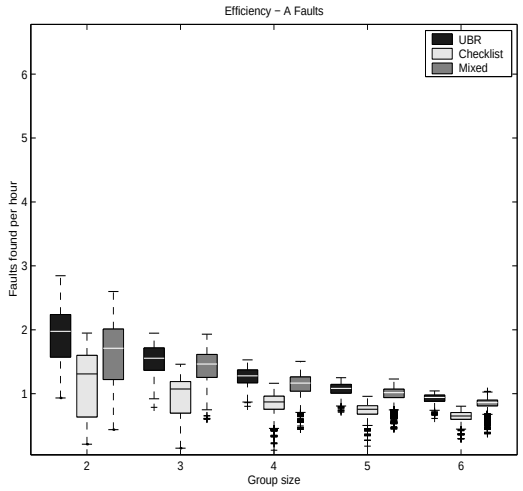

FIGURE 14. Efficiency for different group sizes. Class A faults are included.

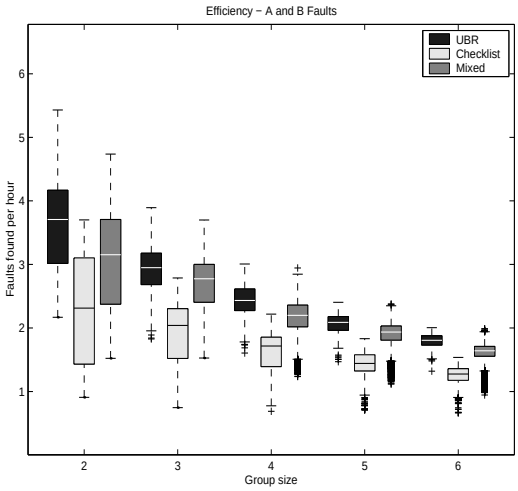

FIGURE 15. Efficiency for different group sizes. Class A and B faults are included.

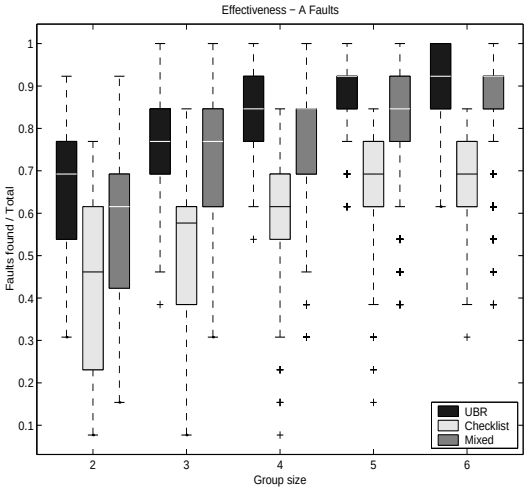

FIGURE 16. Effectiveness for different group sizes. Class A faults are included.

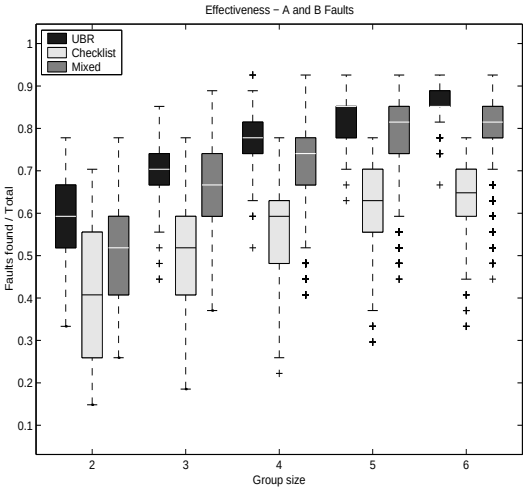

FIGURE 17. Effectiveness for different group sizes. Class A and B faults are included.

8. Discussion

The analysis of the experiment data in the previous section enables rejection of six out of seven null hypotheses, see Table 12.

TABLE 12. Summary of the results of the hypotheses. Statistical significance is marked with an (S), non-significant (–).

Hypothesis / Fault class	All A+B+C	A	A+B
Efficiency	p=0.042 (S)	p=0.013 (S)	p=0.016 (S)
Effectiveness (Rate)	p=0.103 (–)	p=0.036 (S)	p=0.031 (S)
Different (Fault)	p=0.001 (S)	–	–

The result can be interpreted as reviewers using UBR are more efficient and effective than reviewers using CBR. They are significantly more efficient for all faults and for critical-to-user faults. They are more effective for critical faults but not for all faults. The assumption before the experiment was that they would perform better for critical faults, but not necessarily for all faults. The result also shows that reviewers using UBR find different faults compared to reviewers using CBR.

The team performance analysis shows that it is more efficient to use only UBR reviewers. Pure UBR teams are compared to pure CBR teams and also with mixed teams. Although they found different faults, the mixed teams do not outperform the UBR teams. This can be interpreted as reviewers using UBR are so much better that a combination will not help. However, in some cases a small improvement can be observed in terms of effectiveness.

The UBR reviewers started to find faults earlier than the CBR reviewers did. For all faults, an average UBR reviewer started to find faults 20 minutes before an average CBR reviewer. This time increased when critical faults were investigated. This depends on them spending less time to read through the documents. Even if the reviewers using UBR spent less time in both preparation and inspection, they found significantly more faults. The main explanation for this is probably that the use cases helped them to focus on the most important parts of the documents.

The use cases used in UBR were prioritized according to the ranked-based reading method (see Section 3). They were prioritized from a user's perspective, since the purpose was to locate the critical faults from a user's point of view. The checklist was not prioritized according to this principle, since a checklist is not function oriented as is a use case document. However, the checklist items were sorted in a significance order before they were handed out to the subjects. Furthermore, the CBR group had the opportunity to use the exactly same checklist during the introduction to CBR (see Table 6) the day before the actual experiment. The UBR group did not see their use cases before the experiment, since the use cases are different for different systems.

Nevertheless, the experiment shows that the reviewers in the UBR group found significantly more critical faults and performed in total better than the CBR group. The reviewers using CBR found more class C faults. This could depend on the fact when inspecting a document with a checklist, it is easier to focus on details and more difficult to inspect abstract material, which is necessary in order to find severe faults. During the follow-up session, some students said that it would be beneficial to use both the checklist and the use cases. A hybrid of the UBR method would then be beneficial, but we think the checklist would need to be adapted for this purpose in order to use it in combination with the use cases.

The results presented by Thelin et al. [45] show that it is possible to guide reviewers more efficiently and more effectively by prioritizing the use cases in the UBR method. In this experiment, the UBR method is baselined against CBR and the study shows positive results. The results are positive from a research perspective but also from an industrial perspective. Especially software organizations using use cases in their development should be interested in the results.

As always when conducting experiments to increase the body of knowledge, the experiment has to the replicated in different contexts. The replications should address changes in the design, for example, use a different domain and seed more faults into the document under inspection. The method should also be investigated in a case study in an industrial setting in order to evaluate whether it still provides positive effects. It would be especially interesting to investigate the method with professionals as subjects.

In order to develop the method further, an experiment investigating UBR with time-controlled reading should be conducted. If it is possible to control the reviewers' time consumption and thereby focus only on the critical faults, this could be the starting point of a new important reading technique. A hybrid of UBR and CBR should also be investigated. Applying each use case

with some check items or all use cases with a general checklist, the method could be even further improved.

The main purpose of UBR is to focus inspections on users' needs, much in the same way as in usage-based testing (UBT) [38]. Hence, UBR and UBT are two complementary fault detection techniques in the software development, both with the focus on the user of the final product [50]. These two techniques have been compared and the experiment is under analysis. The main purpose of the experiment is to evaluate which of the two techniques that are most effective and efficient and find out what kind of faults they found. For a software organization, the results could lead to a development model with the same general purpose of the verification as well as validation throughout the development.

Several reading techniques exist that are applied in the preparation phase of software inspections. However, no reading technique has been developed for the inspection meeting. Most research has been directed towards improvement of the individual preparation of inspections. Some research studies argue that the meeting is not needed [48]. This may be because the meeting is structured as an ordinary meeting and not as an inspection meeting. To improve the meeting, reading methods for the meeting can be designed to, for example, follow use cases. The meeting should be conducted differently (e.g. role play in design inspections) depending on whether the purpose of the preparation was to detect faults or only to comprehend.

9. Summary

The presented experiment compares two reading techniques in order to baseline the ranked-based usage-based reading method against the standard industry practice of checklist-based reading. UBR showed promising results in an earlier experiment [45] and even more promising results in this experiment.

The main results from the analysis are that reviewers using UBR find more critical faults and do it more efficiently. The fault severity is defined from a user's point of view. The important results from the experiment are:

- Efficiency – Reviewers using usage-based reading are significantly more efficient than reviewers using checklist-based reading. This difference is significant for all faults and for critical faults.
- Effectiveness – Reviewers using usage-based reading are significantly more effective than reviewers using checklist-based reading. This difference is significant for critical faults, but not for all faults.
- Faults – Reviewers using usage-based reading find different and more unique faults and especially more critical faults than reviewers using checklist-based reading.
- Teams – The team analysis also shows that usage-based reading is more effective and efficient than CBR. This is true for all team sizes ranging from two to six reviewers.
- Fault Finding – A reviewer applying usage-based reading starts to find faults earlier than a reviewer using CBR. The differences for all faults are about 20 minutes and this difference is even larger for critical faults.

The further work includes enhancement of UBR, either to include checklist items or to investigate the time-based ranking method. Although the results are promising, the method needs to be replicated and compared with, for example, usage-based testing.

Acknowledgement

The authors would like to thank the students for participating in the investigation and Thomas Olsson at the Department of Communication Systems at Lund University for developing the taxi management system. We would also like to thank Christer Svensson for work on the requirements specification. Thanks also to Dr. Björn Regnell and Johan Natt och Dag at the Department of Communication Systems for prioritizing the use cases and to Dr. Håkan Petersson at the Department of Communication Systems for reviewing this paper. This work was partly funded by The Swedish National Agency for Innovation Systems (VINNOVA), under a grant for the Center for Applied Software Research at Lund University (LUCAS).

References

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

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

[3] Biffl, S., *Software Inspection Techniques to Support Project and Quality Management*, Habilitationsschrift, Shaker Verlag, Austria, 2001.

[4] Bisant, D. B. and Lyle, J. R., "A Two-Person Inspection Method to Improve Programming Productivity", *IEEE Transactions on Software Engineering*, 15(10):1294-1304, 1989.

[5] Ciolkowski, M., Differding, C., Laitenberger, O., and Münch, J., "Empirical Investigation of Perspective-Based Reading: A Replicated Experiment", *ISERN Report* no. 97-13, 1997.

[6] Ebenau, R. G. and Strauss, S. H., *Software Inspection Process*, McGraw-Hill, New York, 1994.

[7] Eick, S. G., Loader, C. R., Long, M. D., Votta, L. G. and Vander Wiel, S., "Estimating Software Fault Content Before Coding" *Proc. of the 14th International Conference on Software Engineering*, pp. 49-65, 1992.

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

[9] Fusaro, P., Lanubile, F., and Visaggio, G., "A Replicated Experiment to Assess Requirements Inspection Techniques", *Empirical Software Engineering: An International Journal*, 2(1):39-57, 1997.

[10] Gilb, T. and Graham, D. *Software Inspections*, Addison-Wesley, UK, 1993.

[11] Halling, M., Biffl, S., Grechenig, T. and Köhle, M., "Using Reading Techniques to Focus Inspection Performance", *Proc. of the 27th Euromicro Workshop on Software Process and Product Improvement*, pp. 248-257, 2001.

[12] Hayes, W., "Research Synthesis in Software Engineering: A Case for Meta-Analysis", *Proc. of the 6th International Software Metrics Symposium*, pp. 143-151, 1999.

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

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

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

[16] Juristo, N. and Moreno, A. M., *Basics of Software Engineering Experimentation*, Kluwer Academic Publisher, USA, 2001.

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

[18] Laitenberger, O. and DeBaud, J-M., "Perspective-Based Reading of Code Documents at Robert Bosch GmbH", *Information and Software Technology*, 39(11):781-791, 1997.

[19]Laitenberger, O. and Atkinson, C., “Generalizing Perspective-based Inspection to Handle Object-Oriented Development Artifacts”, *Proc. of the 21th International Conference on Software Engineering*, pp. 494-503, 1999.

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

[21]Laitenberger, O., Atkinson, C., Schlich, M. and El Emam, K., “An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents”, *Journal of Systems and Software*, 53(2):183-204 2000.

[22]Lanubile, F. and Visaggio, G., “Evaluating Defect Detection Techniques for Software Requirements Inspections”, *ISERN Report* no. 00-08, 2000.

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

[24]Linger, R. C., “Cleanroom Process Model”, *IEEE Software*, 11(2):50-58, 1994.

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

[26]Miller, J., Wood, M. and Roper, M., “Further Experiences with Scenarios and Checklists”, *Empirical Software Engineering: An International Journal*, 3(3):37-64, 1998.

[27]Montgomery, D., *Design and Analysis of Experiments*, John Wiley & Sons, USA, 2000.

[28]Musa, J. D., “Operational Profiles in Software-Reliability Engineering”, *IEEE Software*, 10(2):14-32, 1993.

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

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

[31]Parnas, D. L. and Weiss, D. M., “Active Design Reviews: Principles and Practices”, *Proc. of the 8th International Conference on Software Engineering*, pp. 418-426, 1985.

[32]Petersson, H., Wohlin, C. and Aurum, A., “Effectiveness of Software Inspections as a Guiding Tool to Regulate Team Size”, submitted to *Empirical Software Engineering: An International Journal*, 2002.

[33]Porter, A. and Votta, L., “An Experiment to Assess Different Defect Detection Methods for Software Requirements Inspections”, *Proc. of the 16th International Conference on Software Engineering*, pp.203-112, 1994.

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

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

[36]Regnell, B., Runeson, P. and Thelin, T., “Are the Perspectives Really Different? - Further Experimentation on Scenario-Based Reading of Requirements”, *Empirical Software Engineering: An International Journal*, 5(4):331-356, 2000.

[37]Robson, C., *Real World Research*, Blackwell, UK, 2002.

[38]Runeson, P. and Wohlin, C., “Statistical Usage Testing for Software Reliability Control”, *Informatica*, 19(2):195-207, 1995.

[39]Saaty, T. L. and Vargas, L. G., *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*, Kluwer Academic Publishers, Netherlands, 2001.

[40]Sandahl, K., Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M. and Ohlsson, N., “An Extended Replication of an Experiment for Assessing Methods for Software Requirements”, *Empirical Software Engineering: An International Journal*, 3(4):381-406, 1998.

[41] Shull, F., *Developing Techniques for Using Software Documents: A Series of Empirical Studies*, PhD Thesis, Computer Science Department, University of Maryland, USA, 1998.

[42] Shull, F., Ioana, R. and Basili, V. R., "How Perspective-Based Reading Can Improve Requirements Inspections", *IEEE Computer*, 33(7):73-79, 2000.

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

[44] Sørumgård, S., *Verification of Process Conformance in Empirical Studies of Software Development*, PhD Thesis, Department of Computer and Information Science, The Norwegian University of Science and Technology, Norway, 1997.

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

[46] Thelin, T., Runeson, P., Wohlin, C., Olsson, T. and Andersson, C., "How much Information is Needed for Usage-Based Reading? – A Series of Experiments", *Proc. of the 1st International Symposium on Empirical Software Engineering*, pp.127-138, 2002.

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

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

[49] Weller, E. F., "Lessons from Three Years of Inspection Data", *IEEE Software*, 10(5):38-45, 1993.

[50] Wohlin, C., Regnell., B., Wesslén, A. and Cosmo., H, "User-Centered Software Engineering – A Comprehensive View of Software Development", *Proc. of the Nordic Seminar on Dependable Computing Systems*, pp. 229-240, 1994.

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

[52] Zhang, Z., Basili, V. R. and Schneiderman, B., "Perspective-Based Usability Inspection: An Empirical Validation of Efficacy", *Empirical Software Engineering: An International Journal*, 4(1):43-69, 1999.

Appendix – Checklist for SDL design

Check:

- *Consistency* – The design is consistent if there are no contradictions.
- *Correctness* – The design is correct if it is judged to be compliant with the requirements specification.
- *Completeness* – The design is complete if everything specified in the requirements is also specified in the design.

No.	Where to look	What to check	How to detect	Check when done			
1.	Modules	Consistency	Are the names of the modules consistent?				
2.		Correctness	Are the modules described correctly?				
3.		Completeness	Are all modules included in the design?				
4.	Signals & Parameters	Consistency	Are the names of the signals and parameters consistent?				
5.		Correctness	Is the description of the signals and parameters correct?				
6.		Correctness	Is the number of parameters correct for every signal?				
7.		Correctness	Are the signals specified related to the correct modules?				
8.		Correctness	Is the signal sequencing correct?				
9.		Completeness	Are all signals specified?				
10.	MSC:s	Consistency	Are the MSC:s and the signal specification consistent regarding signals and parameters?				
11.		Correctness	Are all signals included in the MSC:s specified and named correctly?				
12.		Correctness	Is the number of parameters correctly specified?				
13.		Correctness	Is the signal sequencing correct?				
14.		Completeness	Are all modules included?				
15.		Completeness	Are all signal routes specified?				
16.	Introductory text	Consistency	Is the description in the SDL design consistent?				
17.		Correctness	Is the description in the SDL design correct?				
18.		Completeness	Is the description in the SDL design complete?				

Biography

Dr. Thomas Thelin

Dr. Thelin is an associate professor of software engineering at the Department of Communication Systems at Lund University in Sweden. He has a Ph.D. 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. He is a member of IEEE.

Dr. Per Runeson

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 Ph.D. from Lund University in 1998, and a M. Sc. 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 is a member of IEEE.

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

Dr. 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 Linköping University. He has a Ph.D. 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 Science. Dr. Wohlin is on the editorial boards of Empirical Software Engineering: An International Journal and Software Quality Journal. He is a member of IEEE and ACM.