Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment

Reidar Conradi[1], Parastoo Mohagheghi[2], Tayyaba Arif[1], Lars Christian Hegde[1], Geir Arne Bunde[3], and Anders Pedersen[3]

[1] Department of Computer and Information Science, NTNU, NO-7491 Trondheim, Norway
[2] Ericsson Norway - Grimstad, Postuttak, NO-4898 Grimstad, Norway
[3] Agder University College, NO-4876 Grimstad, Norway
conradi@idi.ntnu.no, parastoo.mohagheghi@eto.ericsson.se

Abstract. Object-oriented design and modeling with UML has become a central part of software development in industry. Software inspections are used to cost-efficiently increase the quality of the developed software by early defect detection and correction. Several models presenting the total system need to be inspected for consistency with each other and with external documents such as requirement specifications. Special Object Oriented Reading Techniques (OORTs) have been developed to help inspectors in the individual reading step of inspection of UML models. The paper describes an experiment performed at Ericsson in Norway to evaluate the cost-efficiency of tailored OORTs in a large-scale software project. The results showed that the OORTs fit well into an incremental development process, and managed to detect defects not found by the existing reading techniques. The study demonstrated the need for further development and empirical assessment of these techniques, and for better integration with industrial work practice.

1 Introduction

The Unified Modeling Language (UML) provides visualization and modeling support, and has its roots in object-oriented concepts and notations [4]. Using UML implies a need for methods targeted at inspecting object-oriented models, e.g. to check consistency within a single model, between different models of a system, and between models and external requirement documents. Detected defects may be inconsistencies, omissions or ambiguities; i.e. any fault or lack that degrades the quality of the model.

Typically software inspections include an individual reading step, where several inspectors read the artifacts alone and record the detected defects. An inspection meeting for discussing, classification and recording defects follows this step. Individual reading of artifacts (the target of this paper) strongly relies on the reader's experience and concentration. To improve the output of the individual reading step, checklists and special reading guidelines are provided. Special Object-Oriented Reading Techniques (OORTs) have been developed at the University of Maryland, USA, consisting of seven individual reading techniques (sec. 2.2). In each technique,

either two UML diagrams are compared, or a diagram is read against a Requirements Description.

Modeling in UML is a central part of software development at Ericsson in Grimstad. With increased use of UML, *review* and *inspection* of UML models are done in all development phases. While reviews are performed to evaluate project status and secure design quality by discussing broader design issues, formal inspections are part of the exit criteria for development phases. In the inspection process in Ericsson, individual inspectors read UML diagrams using different views, with checklists and guidelines provided for each type of view or focus.

Ericsson primarily wants to increase the *cost-efficiency* (number of detected defects per person-hour) of the individual reading step of UML diagrams, since inspection meetings are expensive and require participation of already overloaded staff. Ericsson further wants to see if there is any correlation between developer experience and number of defects caught during individual reading. Lastly, Ericsson wants to improve the relevant reading techniques (old or new) for UML diagrams, and to find out whether the new reading techniques fit into their incremental development process.

Before introducing the OORTs in industry, systematic empirical assessments are needed to evaluate the cost-efficiency and practical utility of the techniques. Following a set of student experiments for assessment and improvement of the techniques at The University of Maryland and NTNU [17][6], we conducted a small controlled experiment at Ericsson. The experiment was performed as part of two diploma (MSc) theses written in spring 2002 at the Agder University College (AUC) and The Norwegian University of Science and Technology (NTNU) [5][1]. The original set of OORTs from The University of Maryland were revised twice by NTNU for understandability, evaluated and re-evaluated on two sample systems, and then tailored to the industrial context.

The Ericsson unit in Norway develops software for large, real-time systems. The Requirements Descriptions and the UML models are big and complex. Besides, the UML models are developed and inspected *incrementally*; i.e. a single diagram may be inspected several times following successive modifications. The size of the inspected artifacts and the incremental nature of the software development process distinguish this industrial experiment from previous student experiments. The cost-efficiency of inspections and the types of detected defects were used as measures of the well-suitedness of the techniques. Other steps of the inspection process, such as the inspection meeting, remained unchanged.

Results of the experiment and qualitative feedback showed that the OORTs fit well into the overall inspection process. Although the OORTs were new for the inspectors, they contributed to finding more defects than the existing reading techniques, while their cost-efficiency was almost the same. However, the new techniques ought to be simplified, and questions or special guidelines should be added.

The remainder of the paper is structured as follows: Section 2 describes some state of the art and the new OORTs. Section 3 outlines the overall empirical approach to assess the OORTs. Section 4 summarizes the existing practice of reviews and inspections at Ericsson and some baseline data. Section 5 describes the experimental steps and results, analyzes the main results, and discusses possible ways to improve the new OORTs and their usage. The paper is concluded in section 6.

2 The Object-Oriented Reading Techniques (OORTs)

2.1 A Quick State of the Art

Inspection is a technique for early defect detection in software artifacts [8]. It has proved to be effective (finding relatively many defects), efficient (relatively low cost per defect), and practical (easy to carry out). Inspection cannot replace later testing, but many severe defects can be found more cost-efficiently by inspection. A common reading technique is to let inspectors apply complimentary perspectives or views [2][3]. There are over 150 published studies, and some main findings are:

- It is reported a net productivity increase of 30% to 50%, and a net timescale reduction of 10% to 30% [9, p.24].
- Code inspection reduces costs by 39%, and design inspection reduces rework by 44% [11].
- Ericsson in Oslo, Norway has previously calculated a net saving of 20% of the total development effort by inspection of design documents in SDL [7].

As software development becomes increasingly model-based e.g. by using UML, techniques for inspection of models for completeness, correctness and consistency should be developed. Multiple models are developed for complex software systems. These models represent the same system from different views and different levels of abstraction.

However, there exist no documented, industrial-proven reading techniques for UML-based models [16]. The closest is a reported case study from Oracle in Brazil [13]. Its aim was to test the practical feasibility of the OORTs, but there was no company baseline on inspections to compare with. The study showed that the OORTs did work in an industrial setting. Five inspectors found 79 distinct defects (many serious ones), with 2.7 defects/person-hour (totally 29 person-hours, but excluding a final inspection meeting). Few qualitative observations were collected on how the OORTs behaved.

2.2 The OORTs

As mentioned, one effort in adapting reading techniques for the individual reading step of inspections to object-oriented design was made by the OORT-team at University of Maryland, USA [17]. The principal team members were:

- Victor R. Basili and Jeffrey Carver (The University of Maryland)
- Forrest Shull (The Fraunhofer Center – Maryland)
- Guilherme H. Travassos (COPPE/Federal University of Rio de Janeiro)

Special object-oriented reading techniques have been developed since 1998 to inspect (“compare”) UML diagrams with each other and with Requirements Descriptions in order to find defects. *Horizontal reading techniques* are for comparing artifacts from the same development phase such as class diagrams and state diagrams developed in the design phase. Consistency among artifacts is the most important focus here. *Vertical reading techniques* are for comparing artifacts developed in different development phases such as requirements and design. Completeness

(traceability of requirements into design) is the focus. UML diagrams may capture either *static* or *dynamic* aspects of the modeled system. The original set of OORTs has seven techniques, as in Figure 1:

OORT-1: Sequence Diagrams vs. Class Diagrams (horizontal, static)
OORT-2: State Diagrams vs. Class Descriptions[1] (horizontal, dynamic)
OORT-3: Sequence Diagrams vs. State Diagrams (horizontal, dynamic)
OORT-4: Class Diagrams vs. Class Descriptions (horizontal, static)
OORT-5: Class Descriptions vs. Requirements Descriptions (vertical, static)
OORT-6: Sequence Diagrams vs. Use Case Diagrams (vertical, static/dynamic)
OORT-7: State Diagrams vs. (Reqmt. Descr.s / Use Cases) (vertical, dynamic)

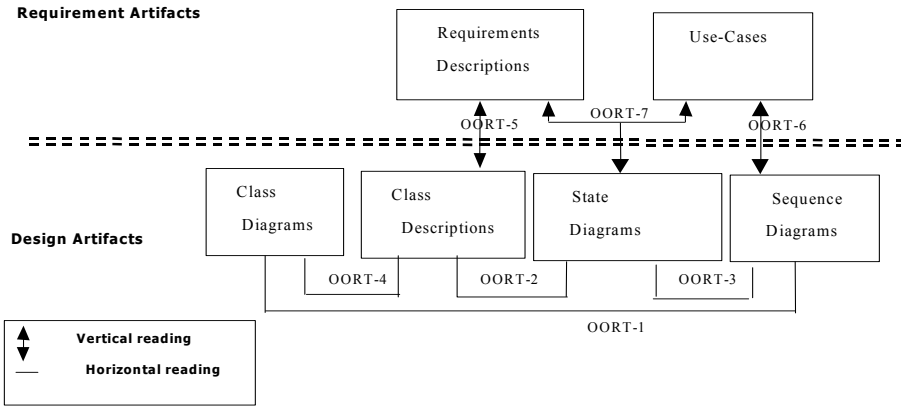

Fig. 1. The seven OORTs and their related artifacts, taken from [18]

The techniques cover most diagrams when modeling a system with UML. In addition, Requirements Descriptions are used to verify that the system complies with the prerequisites. Each technique compares at least two artifacts to identify defects in them (but requirements and use cases are assumed to be defect-free here). The techniques consist of several steps with associated questions. Each technique focus the reader on different design aspects related to consistency and completeness, but not on e.g. maintainability and testability. In student experiments, each reader either did four "dynamic" OORTs or four "static" ones, and with OORT-6 in common. That is, we had two complementary *views*, a dynamic and a static one.

Defects detected by the techniques are classified either as Omission (missing item), Extraneous information (should not be in the design), Incorrect fact (misrepresentation of a concept), Ambiguity (unclear concept), Inconsistency (disagreement between representations of a concept), or Miscellaneous (any other defects). In [18], severity of defects may be either Serious (It is not possible to

[1] Class Descriptions include textual descriptions of goals and responsibilities of a class, list of functions with descriptions of each function, attributes, cardinalities, inheritance, and relations.

continue reading. It needs redesign), Invalidates (the defects invalidates this part of the document) or Not serious (needs to be checked).

To get more familiar with the techniques, a short description of OORT-1 is given in the following: The goal of this technique is to verify that the Class Diagram for the system describes classes and their relationships consistently with the behaviors specified in the Sequence Diagrams. The first step is to identify all objects, services and conditions in the Sequence Diagram and underline them in different colors. The second step is to read the related Class Diagram and see whether all objects are covered, messages and services found, and constraints fulfilled. To help the reader, a set of questions is developed for each step.

3 The Overall Empirical Method

Developing a method solid enough to be used in the industry takes time and effort through various experiments and verification of results. A set of empirical studies at University of Maryland and NTNU has used the empirical method presented in [14] for improving a development process from the conceptual phase to industry. The method is divided into four studies where each study step has some questions that need to be answered before the next level can be reached:

1) Feasibility study -- Did the process provide usable and cost-effective results?
2) Observational study -- Did the steps of the process make sense?
3) Case study: Use in real life cycle -- Did process fit into the lifecycle?
4) Case study: Use in industry -- Did process fit into industrial setting?

Previous studies at The University of Maryland have emphasized steps 1-3, using students. There is also an undocumented student study from University of Southern California, where the OORTs were tailored to the Spiral Model, i.e. step 3. Previous student experiments at NTNU [6] have applied steps 1 and 2.

The mentioned case study at Oracle in Brazil was the *first* industrial study, emphasizing step 4 and feasibility. It applied more or less the original version of the OORTs, i.e. with no tailoring to the industrial context. Regrettably, we were not aware of this study before our experiment.

The study at Ericsson was the *second* industrial study, with emphasis on step 4 and with a direct comparison of Ericsson's existing inspection techniques. It used a revised and tailored version of the OORTs. We will call it an *experimen*t and not a case study, as it was very close to a controlled experiment.

4 The Company Context

The goal of the software development unit at Ericsson in Grimstad, Norway is to build robust, highly available and distributed systems for large, real-time applications, such as GPRS and UMTS networks. SDL and the proprietary PLEX languages have recently been replaced by UML and e.g. Java or C++. UML models are developed to help understanding the structure and behavior of the system, for communicating decisions among stakeholders, and finally to generate code to some extent [10].

The Ericsson inspectors are team members working on the same software system. They have extensive experience with and good motivation for inspections. The artifacts in the student experiments represented complete, although small systems. In contrast, Ericsson's UML models are developed incrementally and updated in each delivery with new or changed requirements. I.e., diagrams are inspected in increments when any complete revision is done. The artifacts at Ericsson are also of industrial calibre:

- The Requirements Descriptions are in many cases large and complex, including external telecommunication standards, internal requirement specifications, and/or change requests.
- The inspected UML diagrams are often huge, containing many classes, relationships or messages - indeed covering entire walls!

4.1 State of the Practice of Reviews and Inspections

Ericsson has a long history in inspecting their software artifacts; both design documents and source code. The inspection method at Ericsson is based on techniques originally developed by Fagan [8], later refined by Gilb [9], adapted for Ericsson with Gilb's cooperation, and finally tailored by the local development department. Below, we describe the existing Ericsson review and inspection process for UML diagrams.

A review is a team activity to evaluate software artifacts or project status. Reviews can have different degrees of formality; i.e. from *informal meetings* (to present the artifacts) and *walkthroughs* (to discuss design issues and whether the design meets the requirements) to *frequent reviews* (more formal intermediate checks for completeness and correctness). Reviews act as internal milestones in a development phase, while formal *inspections* are performed at the end of an activity and act as exit criteria.

Each inspection has an associated team. The team consists of a moderator, several inspectors, at least one author, and possibly a secretary. For optimal performance, Ericsson guidelines state that a team should consist of 5 to 7 persons. The moderator is in charge of planning and initiating the inspection process. He chooses the artifacts to be inspected (with incremental development also their versions), and assigns inspectors to different views (see below). Before the inspection meeting, inspectors individually read the artifacts and mark the defects, usually directly in the inspected artifact. Requirements Descriptions, UML diagrams and source code are usually printed out for easy mark-up. If a diagram is too large to be printed out, the inspector takes separate notes on the defects and related questions.

Ericsson uses *views* during inspections, where a view means to look at the inspected artifact with a special focus in mind. Examples are *requirement* (whether a design artifact is consistent with requirements), *modeling guideline* (consistency with such guidelines), or *testability* (is the modeled information testable?). For each view, the inspectors apply checklists or design rules to help discovering defects.

An example of a modelling guideline is: The interface class will be shown as an icon (the so-called "lollipop") and the connection to the corresponding subsystem, block or unit proxy class shall be "realize" and not "generalize". An example of a design rule is: A call back interface (inherited from an abstract interface) shall be defined on the block or subsystem level (visibility of the interface). Such guidelines

and rules enforce that the design model will contain correct interfaces to generate IDL files.

Only two different classifications for severity of defects are used, Major and Minor. A Major defect (most common) will cause implementation error, and its correction cost will increase in later development phases. Examples include incorrect specifications or wrong function input. A Minor defect does not lead to implementation error, and is assumed to have the same correction cost throughout the whole process. Examples are misspelling, comments, or too much detail.

In spite of a well-defined inspection process and motivated developers, Ericsson acknowledges that the individual reading step needs improvement. For instance, UML orientation is poor, and inspectors spend too little time in preparatory reading - i.e. poor process conformance, see below.

4.2 Inspection Baseline at Ericsson

A post-mortem study of data from inspections and testing was done at the Ericsson development unit outside Oslo, Norway in 1998 [7]. The historical data used in this study is from the period from 1993 to 1998, and also covered data for code reviews and different test activities (unit test, function test, and system test). The results confirm that individual design reading and code reviews are the most cost-efficient (economical) techniques to detect defects, while system tests are the least cost-efficient.

While the cost-efficiency of inspections is reported in many studies, there is no solid historical data on inspection of UML diagrams, neither in the literature nor at Ericsson. As part of a diploma thesis at AUC, data from 38 design and code inspections between May 2001 and March 2002 were analyzed; but note that:

- Design (UML) and code inspections were *not* distinguished in the recorded data.
- In the first 32 inspections logs, only the *total* number of defects was reported, covering both individual reading and inspection meetings. Only the last 6 inspections had distinct data here.

Table 1. Ericsson baseline results, combined for design and code inspections

	%Effort Individual Reading	%Effort Meeting	Overall Efficiency (def./ph)	Individual Reading Efficiency (def./ph)	Meeting Efficiency (def./ph)
All 38 inspections	32	68	0.53	-	-
6 last inspections	24	76	1.4	4.7	0.4

The data showed that most of the effort is spent in inspection meetings, while individual reading is more cost-efficient. For the 6 last inspections:

- 24% of the effort is spent in individual reading, finding 80% of the defects. Inspection meetings took 76% of the effort but detected 20% of defects. Thus, individual reading is 12 times more cost-efficient than inspection meetings.

- Two of these inspections had an extra high number of defects found in individual reading. Even when this data is excluded, the cost-efficiency is 1.9 defects/person-hour for individual reading and 0.6 defects/person-hour for meetings, or a factor 3.

There has been much debate on the effect of inspection meetings. Votta reports that only 8% of the defects were found in such meetings [19]. The data set in this study is too small to draw conclusions, but is otherwise in line with the cited finding.

5 Ericsson Experiment and Results

The experiment was executed in the context of a large, real software project and with professional staff. Conducting an experiment in industry involves risks such as:

- The experiment might be assumed as time-consuming for the project, causing delay and hence being rejected. Good planning and preparation was necessary to minimize the effort spent by Ericsson staff. However, the industrial reality at Ericsson is very hectic, and pre-planning of all details was not feasible.
- The time schedule for the experiment had to be coordinated with the internal inspection plan. In fact, the experiment was delayed for almost one month.
- Selecting the object of study: The inspected diagrams should not be too complex or too trivial for running the experiment. The inspected artifacts should also contain most of the diagrams covered by the techniques.

PROFIT - PROcess improvement For IT industry – is a cooperative, Norwegian software process improvement project in 2000-2002 where NTNU participates. This project is interfaced with international networks on empirical software engineering such as ESERNET and ISERN. For the experiment at Ericsson, PROFIT was the funding backbone.

The OORTs had to be modified and verified before they could be used at Ericsson. Therefore the NTNU-team revised the techniques in two steps:

1. Comments were added and questions rephrased and simplified to improve understandability by making them more concise. The results in [1] contain concrete defect reports, as well as qualitative comments and observations.
2. The set of improved techniques were further modified to fit the company context. These changes are described in section 5.2.

Students experienced that the OORTs were cost-efficient in detecting design defects for two sample systems, as the OORTs are very structured and offer a step-by-step process. On the other hand, the techniques were quite time-consuming to perform. Frustration and de-motivation can easily be the result of extensive methods. In addition, they experienced some redundancy between the techniques. Particularly OORT-5 and OORT-6 were not motivating to use. A lot of issues in OORT-5 and OORT-6 were also covered by OORT-1 and OORT-4. OORTs-6/7 were not very productive either.

The experiment was otherwise according to Wohlin's book [20], except that we do not negate the null hypotheses. The rest of this section describes planning and operation, results, and final analysis and comments.

5.1 Planning

Objectives: The inspection experiment had four industrial objectives, named **O1-O4**:

- **O1 – analyze cost-efficiency and number of detected defects,** with **null hypothesis *H0a:*** *The new reading techniques are as cost-efficient and help to find at least as many defects as the old R&I techniques*. ("Effectiveness", or fraction of defects found in inspections compared to all reported defects, was not investigated.).
- **O2 – analyze the effect of developer experience,** with **null hypothesis *H0b:*** *Developer experience will positively impact the number of detected defects in individual reading.*
- **O3 – help to improve old and new reading techniques for UML**, since Ericsson's inspection guidelines had not been properly updated after the shift in design language from SDL to UML. No formal hypothesis was stated here, and results and arguments are mostly qualitative.
- **O4 – investigate if the new reading techniques fit the incremental development process at Ericsson.** Again, qualitative arguments were applied.

Relevant inspection data: To test the two null hypotheses *H0a* and *H0b*, the *independent* variable was the individual reading technique with two treatments: either the existing review and inspection techniques (R&I) or the OORTs modified for the experiment. The *dependent* variables were the effort spent, and the number and type of detected defects in the individual reading step and in the inspection meetings (see below on defect logs). Data in a questionnaire (from the OORT-team at Maryland) over developer experience was used as a *context* variable. To help to evaluate objectives *O3* and *O4*, all these variables were supplemented with qualitative data from defect logs (e.g. comments on how the OORTs behaved), as well as data from observation and interviews.

Subjects and grouping: Subjects were the staff of the development team working with the selected use case. They were comprised of 10 developers divided in two groups, the *R&I-group* applying the previous techniques and the *OORT-group* applying the new ones. A common moderator assigned the developers to each group. A slight bias was given to implementation experience in this assignment, since Ericsson wanted all the needed views covered in the R&I-group (see however Figure 2 in 5.2). The R&I-group then consisted of three very experienced designers and programmers, one newcomer, and one with average experience. The OORT-group consisted of one team leader with good general knowledge, two senior system architects, and two with average implementation knowledge. Inspection meetings were held as usual, chaired by the same moderator. Since both groups had 5 individuals, the experimental design was balanced. Both groups had access to the same artifacts.

Changes to the OORTs: As mentioned, the OORTs were modified to fit Ericsson's models and documents, but only so that the techniques were comparable to the original ones and had the same goals. The main changes were:

- **Use Case Specifications:** Each use case has a large textual document attached to it, called a Use Case Specification (UCS), including Use Case Diagrams

and the main and alternative flows. This UCS was used instead of the graphical Use Case Diagram in OORT-6 and OORT-7.

- **Class Descriptions:** There is no explicit Class Description document, but such descriptions are written directly in the Class Diagrams. In OORT-2, OORT-4 and OORT-5, these textual class descriptions in the Class Diagrams are used.
- **OORT-4:** Class Diagram (CD) vs. Class Description (CDe). The main focus of this technique is the consistency between CD and CDe. As Class Descriptions are written in the same Class Diagram, this technique seems unnecessary. However, the questions make the reader focus on internal consistency in the CD. Therefore all aspects concerning Class Descriptions were removed and the technique was renamed to "*Class Diagram for internal consistency*".
- **OORT-5:** Class Description (CDe) vs. Requirements Descriptions (RD). Here, the RD is used to identify classes, their behaviors and necessary attributes. That is, the RD nouns are candidates for classes, the RD verbs for behaviors, and so on. The technique was not applicable in Ericsson, due to the large amount of text that should be read. But Ericsson has an iterative development process, where they inspect a small part of the system at one time. The UCS could substitute the RD for a particular part of the system, but the focus of the specification and the level of abstraction demanded major changes in the technique, which would make the technique unrecognizable. Therefore a decision was made to *remove OORT-5*. Thus, we had *six OORTs* to try out.

Defect Logging: To log defects in a consistent and orderly manner, one template was made for the R&I-group and a similar one for the OORT-group – both implemented by spreadsheets. For all defects, the inspectors registered an explanatory name, the associated artifact, the defect type (Omission, Extraneous etc.), and some detailed comments. The OORT-group also registered the technique that helped them to find the defect. Ericsson's categorization of Major and Minor was not applied (we regretted this during later analysis). These changes in defect reporting were the only process modification for the R&I-group. The amount of effort spent by each inspector, in individual reading and inspection meetings, was also recorded for both groups. We also asked for qualitative comments on how the techniques behaved.

5.2 Operation, Quantitative Results, and Short Comments

It was decided to run the experiment in April or May 2002, during an already planned inspection of UML diagrams for a certain use case, representing the next release of a software system. The inspected artifacts were:

- Use Case Specification (UCS) of 43 pages, including large, referenced standards.
- Class Diagram (CD), with 5 classes and 20 interfaces.
- Two Sequence Diagrams (SqD), each with ca. 20 classes and 50 messages.
- One State Diagram (StD), with 6 states including start and stop, cf. below.

Problem note 1: When the actual use case and its design artifacts were being prepared for the experiment, a small but urgent problem occurred: For this concrete

use case (system) there was *no State Diagram (StD)!* Such diagrams are normally made, but not emphasized since no code is generated from these. Luckily, the UCS contained an Activity Diagram that was a hybrid of a StD and a data flow chart. Thus, to be able to use the OORTs in their proposed form, a StD was hastily extracted from this Activity Diagram. However, the StD was now made in the analysis and not in the design phase, so the reading in OORT-7 changed focus. The alternative would have been to drop the three OORTs involving State Diagrams, leaving us with only three OORTs. The R&I-group had access to, but did not inspect this StD.

The experiment was executed over two days. In the beginning of the first day, the NTNU students gave a presentation of the experimental context and setup. For the OORT-group, a short introduction to the techniques was given as well. Since we had few inspectors, they were told to use *all the available six OORTs (excluding OORT-5)*, not just four "dynamic" ones or four "static" ones as in previous experiments (again, we regretted this later).

Each participant filled out a questionnaire about his/her background (e.g. number of projects and experience with UML). The R&I-group was not given any information on the OORTs. The 10 participants in this experiment were the team assigned to the use case, so they had thorough knowledge of the domain and the UML models at hand.

When all participants had finished their individual reading, they met in their assigned teams for normal inspection meetings. During these meetings, each defect was discussed and categorized, and the moderator logged possible new defects found in the meetings as well. At the end of the meetings, a short discussion was held on the usability of the techniques and to generally comment on the experiment.

Problem note 2: One inspector in the OORT-group did only deliver his questionnaire, not his defect log. Thus the OORT-data represents 4, not 5 persons. The number of defects from the OORT group is therefore lower than expected (but still high), while the OORT effort and cost-efficiency data reflect the reduced person-hours.

Table 2. Summary of collected data on defects from the Ericsson experiment

	Indiv. read. defects	Meet. defects	Over-laps	% Indiv. read. defects	% Meet. defects	Person-hours Indiv. read.	Person-hours Meet.
R&I-group	17	8	0	68	32	10	8.25
OORT-group	38	1	8	97	3	21.5	9

Table 2 shows the number of distinctive defects found in individual reading and inspection meetings, both as absolute numbers and relative frequencies. It also shows the effort in person-hours for individual reading and meetings. Defects reported in more than one defect log are called *overlaps* (in column four), and 8 "overlap defects" were reported for the OORT-group.

The cost-efficiency (defects/person-hours) of the individual reading step, the inspection meetings and the average for both groups is shown in Table 3 below.

Table 3. Cost-efficiency of inspections as no. of detected defects per person-hour

	Cost-eff. Indiv.read. (defects/ph)	Cost-eff. Meeting (defects/ph)	Cost-eff. Average. (defects/ph)
R&I-group	1.70	0.97	1.37
OORT-group	1.76	0.11	1.28

Defects logs were used to make a summary of the distribution of defects over the defined defect types. Table 4 shows that the R&I-group registered most Incorrect fact, while the OORT-group found most Omission and Inconsistency.

Table 4. Defect Distribution on defect types

Defect Type	R&I-group Indiv.read.	R&I-group Meeting	OORT-group Indiv.read.	OORT-group Meeting
Omission	3	2	12	1
Extraneous	-	3	6	-
Incorrect fact	10	3	1	-
Ambiguity	-	-	5	-
Inconsistency	2	-	12	-
Miscellaneous	2	-	2	-
Total	17	8	38	1

Short comment: Incorrect facts reported by the R&I-group were mostly detected in the two Sequence Diagrams showing the interactions to realize the use case behavior. These defects were *misuse of a class or interface*, such as wrong order of operation calls or calling the wrong operation in an interface (Incorrect fact was originally defined as misrepresentation of a concept). The group argued that the interface is misrepresented in the Sequence Diagram, and thus the defects are of type Incorrect fact.

For the OORT-group the defects were also classified based on the question leading to find them. OORT-1 and OORT-2 helped finding most defects. OORT-7 did not lead to detection of any defects whatsoever.

Problem note 3: The inspectors mentioned that some defects were detected by more than one technique and only registered the first technique that lead to them. However, the techniques were time-consuming, and one of the developers did not do OORT-6 and OORT-7, while others used little time on these latter two.

As mentioned, the participants filled in a questionnaire where they evaluated their *experience* on different areas of software development on an ordinal scale from 0 to 5, where 5 was best. A total score was coarsely calculated for each participant by simply adding these numbers. The maximum score for 20 questions was 100. Figure 2 shows the number of defects reported by each participant and their personal score for 9 participants (data from the "misbehaving" fifth participant in the OORT-group was not included). The median and mean of these scores were very similar within and between the two groups, so the groups seem well balanced when it comes to experience. For the R&I-group, the number of reported defects increases with their personal score, while there is no clear trend for the OORT- group!

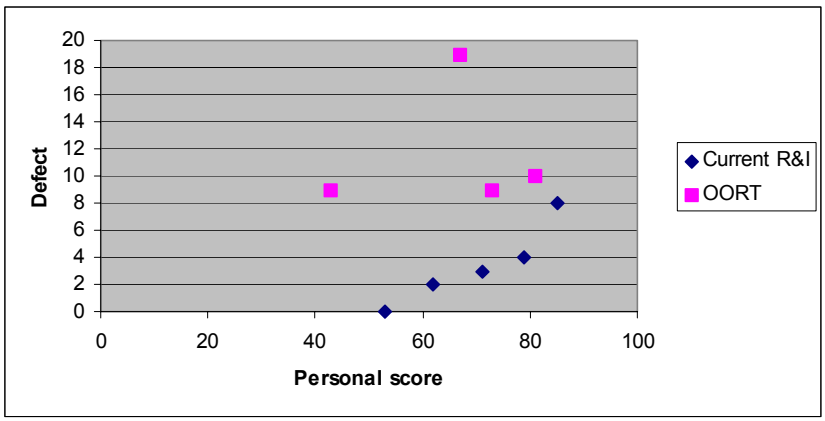

Fig. 2. Relationship between numbers of defects reported in individual reading and personal scores ("experience") for 9 participants

5.3 Further comments, Interpretation and Analysis

Here, we first comment deeper on some of the results, also using qualitative feedbacks. Then we assess the objectives and hypotheses, and lastly analyze the validity threats. A general reminder is that the data material is very meager, so any conclusion or observation must be drawn with great care.

Comments on old vs. new reading techniques: All in all, the R&I-group only found 68% of their defects in individual reading. This is considerably less then the 98% of the defects found by the OORT-group in this step. The meeting was less prosperous for the latter group, which is the expected result. The R&I-group inversely detected 32% of their defects in the inspection meeting, which is high but not cost-efficient. However, the OORT-group spent twice the effort on individual reading, and therefore the cost-efficiency is almost the same. Furthermore, the OORTs were new for the inspectors, and this may hurt cost-efficiency.

The OORT-group found much more Omissions and Inconsistencies than the R&I-group. The OORTs are based on comparing UML diagrams with each other and with requirements, and this may result in finding many more Omissions and Inconsistencies. In contrast, the R&I techniques do not guide inspectors to find "defects", which do not degrade the behavior of the system. An example is possible Inconsistencies between a Class Diagram and a Sequence Diagram (in OORT-1), since no code is generated from a Sequence Diagram during design. However, Inconsistencies in other artifacts related to the State Diagram (as in OORT-2 and OORT-3) are important also for implementation.

The R&I-group detected 10 defects of type Incorrect fact, all being important for implementation, while the OORT-group detected only *one* such defect. The registered defects included both misrepresentation of concepts and misuse of them, such as interface misuse being commented for Figure 4. Finding Incorrect facts may be based

on previous knowledge of the system, and inspectors in the R&I-group had better insight in implementation details. Another reason is, that for the inspected system, internal design guidelines and Class Descriptions contain information on the use of interfaces. Comparing these with the Sequence Diagrams may have helped finding violations to interface specifications, such as wrong order of operation calls. This technique is not currently in the set of OORTs, while the R&I techniques ask for conformance to such design guidelines.

One interesting result of the experiment was the total *lack of overlap* between defects found by the two groups. The N-fold inspection method [12] is based on the hypothesis that inspection by a single team is hardly effective and N independent teams should inspect an artifact. The value of N depends on many factors such as cost of additional inspections and the potential expense of letting a defect slip by undetected. Our results showed that each team only detected a fraction of defects as anticipated by the above method. This result is possibly affected by a compound effect of the two elements discussed earlier as well: slightly different background of inspectors and different focus of reading techniques. The latter meant, that the OORTs focused on consistency between UML diagrams and completeness versus requirements, while the R&I techniques focused on conformance to the internal guidelines. The experiment therefore suggests concrete improvements in the existing R&I techniques.

Lastly, defect severity (e.g. Major, Minor, and possibly Comment or as defined by the OORTs) should be included for both techniques. Defect types might also be made more precise – e.g. to distinguish Interface error, Sequencing error etc.

Comments on the new reading techniques: Some OORTs helped to detect more defects than others. The inspectors mentioned that some defects were found by more than one technique, and were therefore registered only once for the first OORT. Such redundancies should be removed.

Some UML diagrams of the inspected system contain "more" information than others. Modeling is also done differently than assumed in the original set of OORTs - cf. the "Ericsson" changes to OORT-4 and removal of OORT-5.

As mentioned, for the inspected system we had to improvise a State Diagram from an Activity Diagram already standing in the Use Case Specification. But again, making an explicit and separate State Diagram proved that the new OORTs really work: 16(!) defects were totally identified using OORT-2 and OORT-3, comparing the State Diagram with, respectively, Class Descriptions and Sequence Diagrams.

The participants in the OORT-group said it was too *time-consuming* for each to cover all the OORTs, and some (often the last) techniques will suffer from lack of attention. A possible solution is to assign only a subset of the techniques to each participant, similarly to Ericsson's *views* and to what was done in earlier student experiments. A more *advanced UML editor* might also catch many trivial inconsistencies, e.g. undefined or misspelled names, thus relieving human inspectors from lengthy and boring checks.

Finally, we should *tailor* the reading techniques to the context, i.e. project. For instance, the OORTs were successful in detecting Omissions and Inconsistencies by comparing UML diagrams with each other and with the requirements. But they did not detect e.g. misuse of interfaces and inconsistencies between the models and the internal guidelines. A natural solution is to include questions related to *internal*

guidelines and design rules, and then e.g. compare Sequence Diagrams with class and interface descriptions as part of a revised OORT-1.

Evaluation of O1/H0a – cost-efficiency and number of defects: Our small sample prevents use of standard statistical tests, but we can anyhow assess *H0a* (and *H0b* below). The cost-efficiency of the old and new techniques seems rather similar, and in line with that of the baseline. The OORTs seem to help finding more defects in the individual reading step than the R&I techniques, respectively 38 and 17 defects. Even without defects (indirectly) related to the new State Diagram, 22 defects were reported using the OORTs. Thus the null hypothesis *H0a should be accepted*.

Evaluation of O2/H0b – effect of developer experience on number of defects: From Figure 2 we see that the number of reported defects from the individual reading step increases with the personal score for the R&I-group. This may indicate that the R&I techniques rely on the experience of the participants. But there is no clear relationship for the OORT-group. Thus the null hypothesis *H0b should be accepted for the R&I-group,* but we cannot say anything for the OORT-group. The effect for the OORT-group is surprising, but consistent with data from The University of Maryland and NTNU [6], and will be documented in Jeffrey Carver's forthcoming PhD thesis.

Evaluation of O3 – improvement of reading techniques for UML: The new OORTs helped Ericsson to detect many *defects not found* by their *existing* R&I techniques. However, both the old and new reading techniques varied a lot in their effectiveness to detect defects among different diagrams and diagram types. This information should be used to improve both sets of reading techniques. Actually, there were many comments on how to improve the OORTs, suggesting that they should be shortened and simplified, have mutual redundancies removed, or include references to internal design guidelines and rules. Thus, although the original set of OORTs had been revised by NTNU in several steps and then tailored for Ericsson, the experiment suggests further simplification, refinement, and tailoring.

Evaluation of O4 – will fit in the incremental development process: Although the OORTs were originally created to inspect entire systems, they work well for an *incremental* development process too. The techniques helped to systematically find inconsistencies between new or updated UML diagrams and between these diagrams and possibly changed requirements. That is, they helped inspectors to see the revised design model as a whole.

Validity Evaluation: Threats to experimental validity are classified and elaborated in [15] [20]. Threats to validity in this experiment were identified to be:

- *Internal validity*: There could be some compensatory rivalry; i.e. the R&I-group could put some extra effort in the inspection because of the experiment. Inversely, the OORT-group may do similar in a "Hawthorne" effect. Due to time/scheduling constraints, some participants in the OORT-group did not cover all the techniques properly, e.g. OORT-6 and OORT-7.
- *External validity*: It is difficult to generalize the results of the experiment to other projects or even to other companies, as the experiment was done on a single use case. Another threat was that the OORTs were adapted for Ericsson, but we tried to keep the techniques as close to the original set as possible.
- *Construct validity*: The OORT-group had knowledge of the R&I techniques and the result for them could be a mix of using both techniques.

- *Conclusion validity*: The experiment is done on a single use case and it is difficult to conclude a statistical relationship between treatment and outcome. To be able to utilize all the techniques, a simple State Diagram was extracted the day before the experiment. The R&I-group did not look at this particular diagram, while the OORT-group reported 16 defects related to this diagram and to indirectly related artifacts. The inspectors were assigned "semi-randomly" to the two groups, which roughly possessed similar experience. The adding of ordinal scores to represent overall inspector experience is dubious, but this total score was only used qualitatively (i.e. is there a trend? - not how large it is).

6 Conclusions

The studied Ericsson unit incrementally develops software for large-scale real-time system. The inspected artifacts, i.e. Requirements Descriptions and UML models, are substantially larger and more complex than those used in previous academic experiments. For Ericsson it is interesting to see if these techniques could be tailored to their inspection needs in the individual reading step.

Below we sum up the objectives of the experiment and how they have been reached:

- **O1 and H0a – cost-efficiency and detected defects:** The cost-efficiency of the old R&I techniques and the new OORTs seems very similar. The new ones helped to find more than twice as many defects as the old ones, but with no overlaps with the defects found by the old techniques.
- **O2 and H0b – effect of developer experience on detected defects:** There is probably a positive trend for the old R&I techniques, but we do not know for the new ones. The result may term "expected", but the reasons are not quite understood.
- **O3 - improvement of old and new reading techniques:** Although the new OORTs have shown promising results, the experiment suggests further modifications of both general and specific issues. We have for both the old and the new reading techniques identified parts that could be included in the other.
- **O4 – fit into an incremental process:** To our surprise this went very well for the OORTs, although little attention and minimal effort was spent on this.

To conclude: In spite of very sparse data, the experiment showed a need for several concrete *improvements*, and provided many unforeseen and valuable *insights*. We also should expect a learning effect, both for the reading techniques and for Ericsson's inspection process and developers, as a result of more OORT trials. We further think that the evaluation process and many of the experimental results can be *reused* in future studies of inspections of object-oriented design artifacts in UML.

Some final challenges: First, how to utilize inspection data actively in a company to improve their inspection process? Second, how to convince the object-oriented community at large, with its strong emphasis on prototyping and short cycle time, to adopt more classic quality techniques such as inspections?

Acknowledgements

We thank Ericsson in Grimstad for the opportunity and help to perform the experiment with ten of their designers and several managers, who all were highly motivated. We also thank the original OORT-team in USA for inspiration and comments. The study was partially funded by two public Norwegian research projects, namely PROFIT (sec. 5) and INCO (INcremental and COmponent-based development, done jointly by University of Oslo and NTNU). Thanks also goes to local colleagues at NTNU.

References

1. Arif, T., Hegde, L.C.: Inspection of Object-Oriented Construction. Diploma (MSc) thesis at NTNU, June 2002. See http://www.idi.ntnu.no/grupper/su/su-diploma-2002/Arif-OORT_Thesis-external.pdf.
2. Basili, V.R., Caldiera, G., Lanubile, F., and Shull, F.: Studies on reading techniques. Proc. Twenty-First Annual Software Engineering Workshop, NASA-SEL-96-002, p. 59-65, Greenbelt, MD, Dec. 1996.
3. Basili, V.R., Green S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M. V.: The Empirical Investigation of Perspective-Based Reading, *Empirical Software Engineering Journal, 1(2):*133-164, 1996.
4. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, 1999.
5. Bunde, G.A., Pedersen, A.: Defect Reduction by Improving Inspection of UML Diagrams in the GPRS Project. Diploma (MSc) thesis at Agder University College, June 2002. See http://siving.hia.no/ikt02/ikt6400/g08/.
6. Conradi, R.: Preliminary NTNU Report of the OO Reading Techniques (OORT) exercise in course 7038 on Programming Quality and Process Improvement, spring 2000, v1.12. Oct. 2001, 80 p.
7. Conradi, R., Marjara, A., Hantho, Ø., Frotveit, T., Skåtevik, B.: A study of inspections and testing at Ericsson, Norway. Proc. PROFES'99, 22-24 June 1999, p. 263-284, published by VTT.
8. Fagan, M. E.: Design and Code Inspection to Reduce Errors in Program Development. *IBM Systems Journal, 15 (3):182-211*, 1976.
9. Gilb, T., Graham, D.: Software Inspection. Addison-Wesley, 1993.
10. Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, revised printing, 1995.
11. Laitenberger, O., Atkinson, C.: Generalized Perspective-Based Inspection to handle Object-Oriented Development Artifacts. Proc. ICSE'99, Aug. 1999, IEEE CS-Press, p. 494-503.
12. Martin, J., Tsai, W.T.: N-fold Inspection: A Requirements Analysis Technique. *Communications of the ACM, 33(2)*: 225-232, 1990.
13. Melo, W., Shull, F., Travassos, G.H.: Software Review Guidelines, Technical Report ES-556/01, Aug. 2001, 22 p. Systems Engineering and Computer Science Department, COPPE/UFRJ, http://www.cos.ufrj.br (shortly reporting OORT case study at Oracle in Brazil).
14. Shull, F., Carver, J., Travassos, G.H.: An Empirical Method for Introducing Software Process. Proc. European Software Engineering Conference 2001 (ESEC'2001), Vienna, 10-

14 Sept. 2001, ACM/IEEE CS Press, ACM Order no. 594010, ISBN 1-58113-390-1, p. 288-296.
15. Sommerville, I.: Software Engineering. Addison-Wesley, sixth ed., 2001.
16. Travassos, G.H., Shull F., Carver J., Basili V.R.: Reading Techniques for OO Design Inspections, Proc. Twenty-Forth Annual Software Engineering Workshop, NASA-SEL, Greenbelt, MD, Dec. 1999, http://sel.gsfc.nasa.gov/website/sew/1999/program.html.
17. Travassos, G.H., Shull, F., Fredericks, M., Basili, V.R.: Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. Proc. OOPSLA'99, p. 47-56, Denver, 1-5 Nov. 1999 (in *ACM SIGPLAN Notices,34(10)*, Oct. 1999).
18. Travassos, G.H., Shull, F., Carver, J., Basili, V.R.: Reading Techniques for OO Design Inspections. University of Maryland Technical Report CS-TR-4353. April 2002 (OORT version 3), http://www.cs.umd.edu/Library/TRs/CS-TR-4353/CS-TR-4353.pdf.
19. Votta, L.G.: Does Every Inspection Need a Meeting? Proc. ACM SIGSOFT'93 Symposium on Foundation of Software Engineering (FSE'93), p 107-114, ACM Press, 1993.
20. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering, an Introduction. Kluwer Academic Publishers, 2000.