Defect Reduction by improving Inspection of UML Diagrams in the GPRS Project

---

A study of available techniques and "state-of-the-practice" in the GPRS project for inspection of UML diagrams. Experimenting a suggested method to reduce defect cost by early detection of defects.

IKT6400 – Diploma Thesis

Siv.ing. degree in
Information and Communication
Technology

June, 2002

Geir Arne Bunde & Anders Pedersen

ERICSSON

Abstract

Since UML was introduced to the software development industry, companies have adopted this notation language into their Object-Oriented development process.

Our object in this thesis has been to evaluate the current R&I method against new reading techniques for object oriented diagrams and models. The new techniques, OORT focuses the inspectors in a development process on the model and help finding defects of different character than their current R&I method.

We preformed an experiment at Ericsson together with students at NTNU, using the new techniques. Results from it shows that the OORT's focuses the inspectors in a development process on the Model and help finding defects of different character than their current R&I method. The techniques also lead the inspector to find more subtle defects. Ericsson's current R&I method found more defects of technical value. This makes the two techniques complementary.

We found that the OORT inspections would fit into RUP, if the architecture-centric approach is used. This approach sees the system as an entity, possible divided into several sub-entities. Each entity is self-contained with a set of information that is conceptually whole and logically complete.

OORT-inspections are not restricted to a deadline, since it is not performed because of implementation of a new functionality. When a system entity's functionality has been fully or partly covered up, an OORT inspection can be performed whenever, after this level has been reached.

As far as we can see from the results, Ericsson would profit from implementing OORT in their inspection process. Further on, guidelines for the new technique must be developed and more industrial experiments should be performed with the OORT's. The OORT's themselves should also be generalized to fit modern Object-Oriented Development Processes as RUP, Extreme Programming and others.

Preface

This thesis is written for Ericsson, Grimstad and is performed to complete the Master of Science degree in Information and Communication Technology (ICT) at Agder University College, Faculty of Engineering and Science in Grimstad, Norway.

This thesis is a part of the R&D programme "The Mobile Student".

During the project time we were so lucky to get access to the Ericsson file system. We had access to the GPRS model and process documentation to study the current inspection techniques. Parts of the time we had our own workstation at Ericsson, which were very helpful to us.

We would like to thank the GPRS project consented in doing the experiment and PROFIT [1] (Process Improvement for IT industry) for financing the project, so we were able to perform the experiment at Ericsson. We would like to thank Gunhild S. Lundvall for her work in making this possible, Bjørn E. Jensen's Team, who where the participants of the experiment. We want to thank Reidar Conradi for his involvement, and Lars Christian Hegde and Tayyaba Arif for their cooperation and valuable work with the new inspection techniques.

We would also like to thank Parastoo Mohagheghi at Ericsson for her guidance and valuable help, Jan P. Nytun at HiA for his interest in the project and his follow up during the process and Stein Bergsmark at Ericsson for his engagement to make us work harder.

Geir Arne Bunde Anders Pedersen

Grimstad, Norway June 2002

Contents

1 Introduction

1.1 Thesis introduction

Since UML was introduced to the software development industry, companies have adopted this notation language into their Object-Oriented development process. In Ericsson's case this resulted in a move from SDL to UML [2].

Software inspections are a proven concept that is much used in industry. Some defects can not be found by testing, and defects found late are expensive to correct. Detecting defects on an early stage in the development process will reduce the costs on rework considerably. Thus techniques for early defect detection are needed. Software inspections in general have not been focused on inspecting UML-diagrams, but rather textual documents. The reading techniques used when inspecting such documents are becoming less and less relevant for use with the Object-Oriented paradigm.

The GPRS project at Ericsson takes advantage of the Rational Unified Process [3] and UML diagrams for requirement engineering, analysis, design and test. While the GSN RUP adaptation describes the artifacts that should be produced in different stages of the project life cycle, inspection of artifacts has received less attention. Ericsson's current R&I method, describes *what* defects to look for but not *how* to find them.

The new reading techniques that are evaluated in this Thesis, OORT (Object-Oriented Reading Techniques), are based on traceability between diagrams/documents. These reading techniques consist of procedural guidelines, comparing two or three diagrams/documents with each other in order to find defects. OORT are in the early stages, and industrial experiments are needed to find out if they fit into modern software development processes.

During our thesis we have performed an experiment at Ericsson, comparing the two techniques, giving a qualitative feedback on the usability of the new techniques. The result of this thesis will be used for product and process improvement in the organization and in association with NTNU, be part of PROFIT experiments to learn how to conduct studies, to be able to react to challenges in the future. This thesis intend to give Ericsson valuable input on how to improve their inspection process and how they could tailor a new reading technique to be used with UML design.

During our work, we learned more about OO design, inspection techniques and goals, the Rational Unified Process, performing experimental studies in the industry and analysing the results.

1.2 Work/Task description

Thesis Title: Defect reduction by Improving Inspection of UML diagrams in the GPRS project.

Subtitle: A study of available techniques and state-of-the-practice in the GPRS project for inspection of UML diagrams. Experimenting a suggested method to reduce defect cost by early detection of defects.

We worked towards the thesis goals during the thesis:

- ? Study the inspection techniques for UML diagrams.
- ? Study the state-of-the-practice in the GPRS project for inspection of UML diagrams.
- ? Design and conduct an experiment where the subject is to compare the existing inspection technique in the GPRS project for UML diagrams with an assumed improved variant.
- ? Based on the studies and the experiment results, suggest improvements to the inspection techniques for UML diagrams in the GPRS project.
- ? Develop guidelines that may be used by reviewers during inspections.

The experiment is based on results from a pre-diploma thesis written at NTNU and an experiment done during spring 2002 where the participants are students from a course at NTNU. We cooperated with he students from NTNU with the design of these experiments. The goal for us was to learn how to conduct and analyse industrial experiments and to evaluate the suggested improvement. Participants of the experiment were employees at Ericsson.

1.3 Literature review

This section explains where information we have studied, and what is relevant for this thesis can be found. For more specific information about articles, see the reference, chapter 10.

To be able to get an overview of the status of object oriented modeling and reading techniques, we have studied a number of articles, which most of them where public. Relevant background material can be found in articles written by Travassos, Basilli, Carver and Shull.

When we where to study the baseline at Ericsson, we had to find material concerning the current method for review and inspections. A lot of the baseline material was found at the Ericsson intranet, which is not public. But the materiel there is based in articles and books about RUP [3], UML [2] and software inspection.

Concerning the experiment preformed in this thesis, we studied various papers about experimenting and experimenting design. A lot of information about experimentation can be found in the book “Experimentation in software engineering”, by Wholin and others [5]. It presents an introduction to experimentation.

When it comes to the new techniques introduced in the experiment we based our studies on the work with the OORT’s that was preformed by the NTNU students, and their previous work with the technique, it can be found in their pre diploma thesis [6].

1.4 Report outline

Since this is an assignment for Ericsson, to improve the inspection routines, the target group for this report is employees at Ericsson. Since we started to work with the students at NTNU, there is also interesting for people working with software inspections, to read this report. Other target groups can be students and engineers with basic knowledge or/and interest of object oriented reading techniques.

First, chapter 2 is background information to introduce Object Oriented Reading Techniques (OORT). Where we explain the background of OORT and what it is.

The next chapter explains the state of art at Ericsson today. What their routines are today and what the inspection process is. We also sat in at an inspection meeting, which is described in detail. Which is followed by a chapter explaining the process before an experiment, it includes e.g. different threats in an experiment.

Then the actual experiment is presented. This is the practical part of the thesis, which explains our process, the experiment, the results and an analysis of the results. Which is followed by the most important chapter, at least for Ericsson, it explains our suggested improvements which can be implemented at Ericsson, to use in their inspection routines.

And finally the conclusion; is this the way to go for Ericsson or are there are any other methods to use.

2 Object Oriented Reading Techniques

2.1 Background/history

The reading techniques were developed by Travassos, Shull and Carver at the University of Maryland. The background was that while developers are usually taught how to write software documents, the skills required for effective reading are rarely taught and must be built up through experience. Reading software documents is also a key industrial activity and research in this area seemed to be valuable.

The first experiment was conducted at the University in Maryland. The result was presented in OOPSLA [7] from a conference in Denver, 1999.

There has been ongoing experiments and work with the reading techniques at University of Maryland. There was conducted a survey at Oracle, Brazil, in 2001. This was the first time for checking the techniques in an industrial setting. The results (though much of them was held back by the company), can be found in COPPE01 [8].

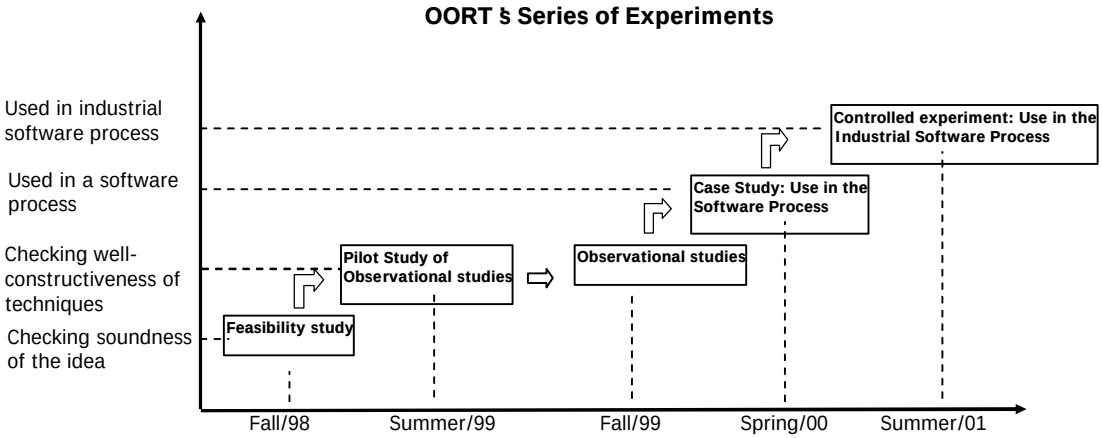


Figure 2.1 – OORT's series of experiments

There has been done some important work at NTNU by Reidar Conradi and students Tayyaba Arif and Lars Christian Hedge. The first academically experiment was accomplished in year 2000 by Conradi with students as subjects. The techniques were performed on two example system designs. See Conradi01 [9] for more details. The result of these experiments was concrete discrepancy reports from the reading techniques, but the students also gave general comments. The comments given covered both the software artifacts used in the experiment to test the reading techniques, and general comments on the techniques themselves. Later on the two mentioned students performed a pre-diploma study, with a quasi-experiment on a set of the techniques that had been structured by Reidar Conradi. The two students made some suggestions for improvements and also conducted another academical experiment [6], at NTNU, based on improvements with some adaptations to the use at Ericsson.

The experiment at Ericsson in our diploma was preformed in cooperation with these two students and Reidar Conradi. This is the first industrial experiment that compares an existing model inspection method with the new OORT's. The experiment will fit into ISERN definition of a "level 3" experiment [10].

2.2 Reading Techniques (OORT)

2.2.1 Object oriented design

Before we learn about OORT we have to understand what Object-Oriented Design is. SEW99 [4] describes it as "a set of diagrams concerned with the representation of real world concepts as a collection of discrete objects that incorporate both data structure and behavior."

We differ between low-level and high-level design. High-level design is made after requirements document(s) are finished. High-level design captures the requirements and gives them a new graphical notation in an attempt to give developers understanding of the problem. High-level design does not try to solve the problem, but that is the case with Low-level design.

Low-level design is a model for the code. We can ensure higher quality for these diagrams by inspecting the High-level design, since Low-level design uses the same set of models. This will be beneficial for the (software) coders.

The set of Reading Techniques that was developed in Maryland was concerned with UML notation and the following diagrams: class, interaction (sequence and collaboration), state machine and package. When used at Ericsson we had to adapt the techniques to include the models that *they* used in high-level design.

The original set is based on the following sources for defining High-level design [4] (SEW99):

- A set of functional requirements that describe the concepts and services that are necessary in the final system;
- Use cases that describe important concepts of the system (which may eventually be represented as objects, classes, or attributes) and the services it provides;
- A class diagram (possibly divided into packages) that describes the classes of a system and how they are associated;
- A set of class descriptions that list the classes of a system along with their attributes and behaviors;
- Sequence diagrams that describe the classes, objects, and possibly actors of a system and how they collaborate to capture services of the system;
- State diagrams that describe the internal states in which a particular object may exist, and the possible transitions between those states.

The figure below shows an OO software process:

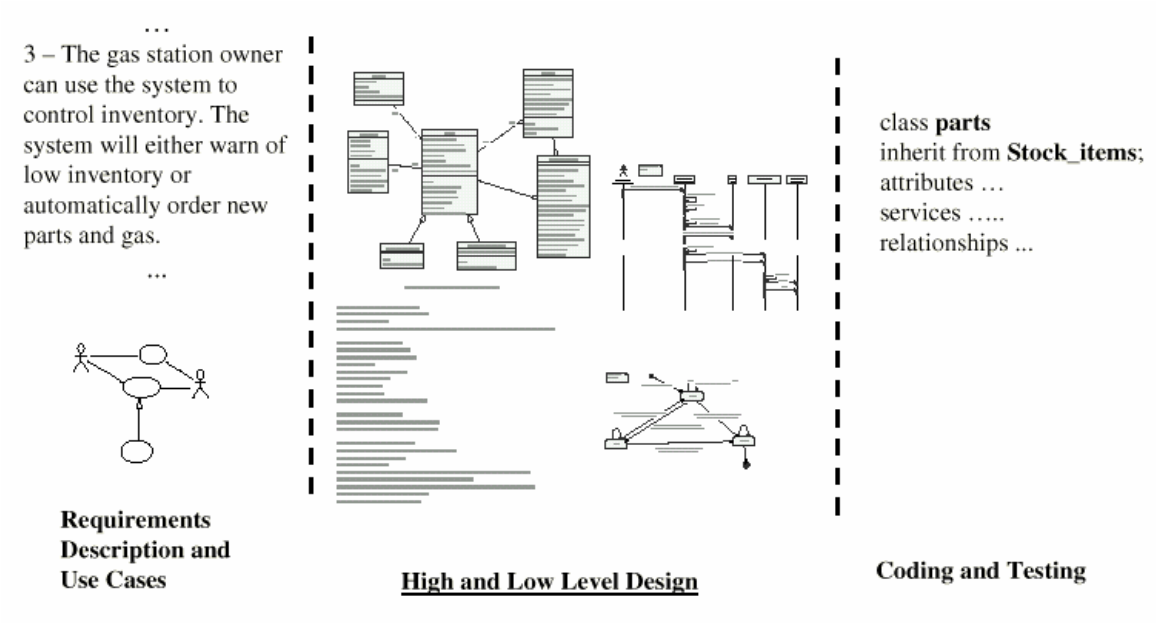


Figure 2.2 – Object oriented software process

2.2.2 Horizontal and vertical reading

The reading techniques are sets of procedural guidelines that can be followed step-by-step by the inspectors, when going through the diagrams that accord to the specific reading technique. The Reading techniques are namely divided into seven different techniques, each concerned about a specific view on two or tree diagrams. These are diagrams that are beneficial to compare with each other.

The main focus is to find defects. They were grouped into vertical or horizontal reading techniques, where Vertical is reading of diagrams over different life cycle phases and horizontal is in the same life cycle phase. See the figure 2.3.

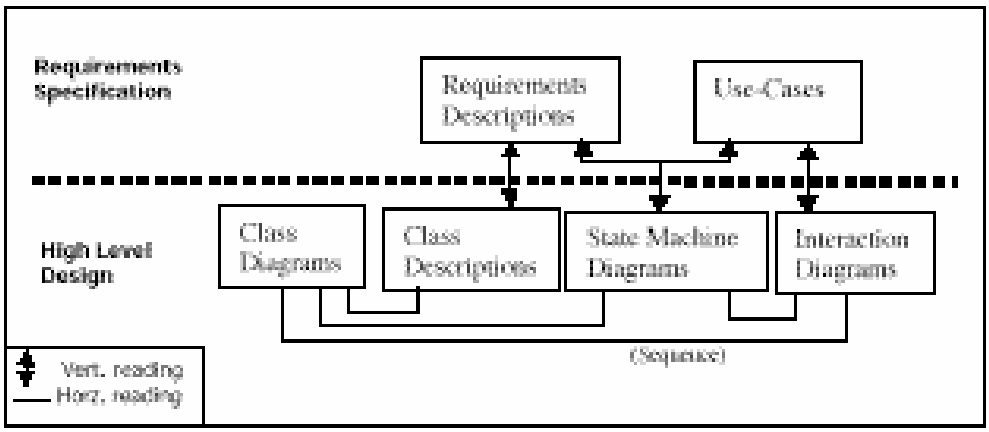


Figure 2.3 – Traceabilty between artifacts

A nice way to put the difference between horizontal and vertical reading is that horizontal aim to identify whether all of the design artifacts are describing the same system, and that vertical reading tries to verify whether those design artifacts represent the right system, described by the requirements and use-cases [4]. It is not necessary to use all techniques and it is not necessary to follow the order presented. Although it seems reasonable to use horizontal techniques first, to ensure that a consistent system will be checked against requirements.

Since the level of abstraction in the requirements is different than those in the design artifacts, it could be a help to divide system functionality into three parts: Messages, Services and Functionality. Messages represent the communication between objects that work together to implement system behavior. Services are combinations of one or more messages and usually capture some basic activity necessary to accomplish functionality. Functionality is what the end-user expects to be visible.

There are two other important terminologies that tell how the functionality is to be implemented, not only what. These are constraints and conditions. A condition describes what must be true for the functionality to be executed. A constraint must always be true for system functionality.

Why should we perform horizontal reading?

UML organizes their artifacts based on the perspective it's capturing system information. Some of the artifacts capture static information. That is; the structure assumed by objects of the domain, when playing specific roles in the problem domain. Other artifacts capture dynamic information. That is; the consequences when objects are asked to perform certain tasks to accomplish system functionalities. In order to understand whether all these artifacts represent the same system, we apply horizontal reading.

Horizontal reading covers the semantic gap between artfiacts. I.e. the differences between a sequence diagram and a state diagram, where sequence diagram shows messages sent between objects and the state diagram show how objects *react* to messages, services or functionality.

Why should we perform vertical reading?

There is no separation of concerns, nor a direct mapping between the two phases. Vertical reading helps the reader identify the information he or she is looking for. I.e. a sequence diagram are organized based on messages that that work together to provide services that compose the right functionality. The information that the designers base these decisions upon comes from Requirements and Use-cases which do not contain messages, but only functionality and in some cases services. In a sequence diagram that information must be made explicit and associated with the messages. Vertical reading explores these differences and helps the reader find faults specific types (see defect-taxonomy in next subchapter).

Are there other benefits except from those of software quality? Yes, there are economical benefits. Conradi99 [3] claims that Design Inspections tend to catch 2/3 of the defects before testing, by spending 10% of the development effort and thereby saving about 20% of the total effort (by earlier defect correction).

2.2.3 Brief description of the techniques

The techniques (complete set can be found in Appendix C):

- OORT-1 Sequence Diagram x Class Diagram (Horizontal, Static)
- OORT-2 State Diagram x Class description (Horizontal, Dynamic)
- OORT-3 Sequence Diagram x State Diagram (Horizontal, Dynamic)
- OORT-4 Class Diagram x Class Description (Horizontal, Static)
- OORT-5 Class Description x Requirements description (Vertical, Static)
- OORT-6 Sequence Diagram x Use Case Diagram (Vertical, Dynamic/Static)
- OORT-7 State Diagram x (Requirement Description and Use Case)(Vertical, Dynamic)

Whether the techniques are static or dynamic is open for discussion.
According to the figure shown earlier, the techniques are applied in this way:

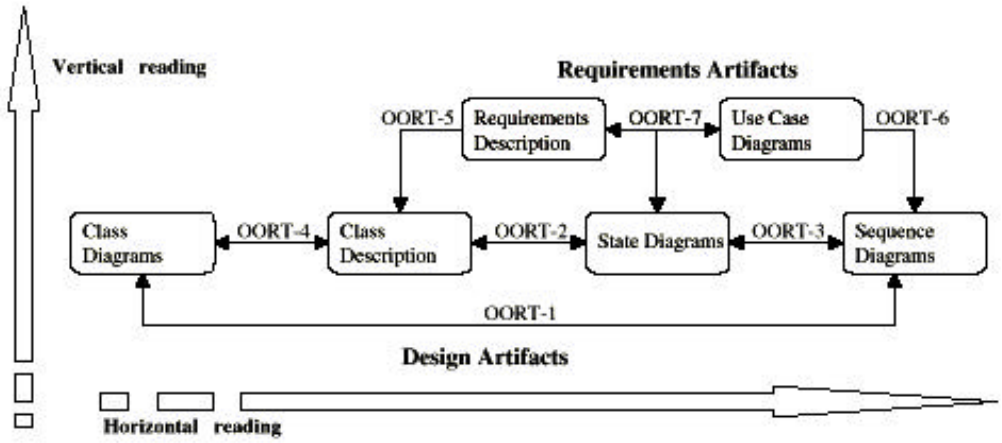


Figure 2.4 - Object-Oriented Reading Techniques, OORT

Requirements and Use-Case diagrams are input documents to the process. They are not inspected for discrepancies, but serve as a reference for the other documents.
Requirements inspection can be done by using PBR (Perspective Based Reading).
Each of the seven techniques has listed input and output documents, goals and instructions on how to reach them. There is also included some examples.

In the paper "Reading Techniques for OO Design Inspections" [4] it is defined the following defect taxonomy that the reading techniques are based on:

Type of defect	***Description***
Omission	One or more design diagrams that should contain some concept from the general requirements or from the requirements document do not contain a representation for that concept.
Incorrect Fact	A design diagram contains a misrepresentation of a concept described in the general requirements or requirements document.
Inconsistency	A representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another design diagram.
Ambiguity	A representation of a concept in the design is unclear, and could cause a user of the document (developer, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept.
Extraneous Information	The design includes information that, while perhaps true, does not apply to this domain and should not be included in the design.

Table 2.1 – Defect taxonomy

2.3 Reading Techniques (PBR)

The basic goal of PBR is to examine the documentation of a logical entity from the perspectives of the entity's various stakeholders. An inspector using PBR therefore reads the documentation from the perspective of a particular stakeholder in such a way as to determine whether it satisfies the stakeholders' particular needs. A stakeholder perspective may be, for example, a future user of the system who wants to ensure the completeness of the inspected analysis documents.

During the reading process, an inspector follows the instructions of a perspective-based reading scenario (in short: scenario). A scenario tells the inspector how to go about reading the documentation from one particular perspective and what to look for. A scenario consists of an introduction, instructions, and questions framed together in a procedural manner. Oliver Laitenberger [11] suggests that PBR are not only used on textual documents, but also on graphical notations as UML diagrams.

Once an inspector has achieved an understanding of the documented information related to the logical entity, he or she can examine and judge whether this documentation fulfils the required quality properties. For making this judgment an inspector is supported by a set of questions that are answered while following the instructions.

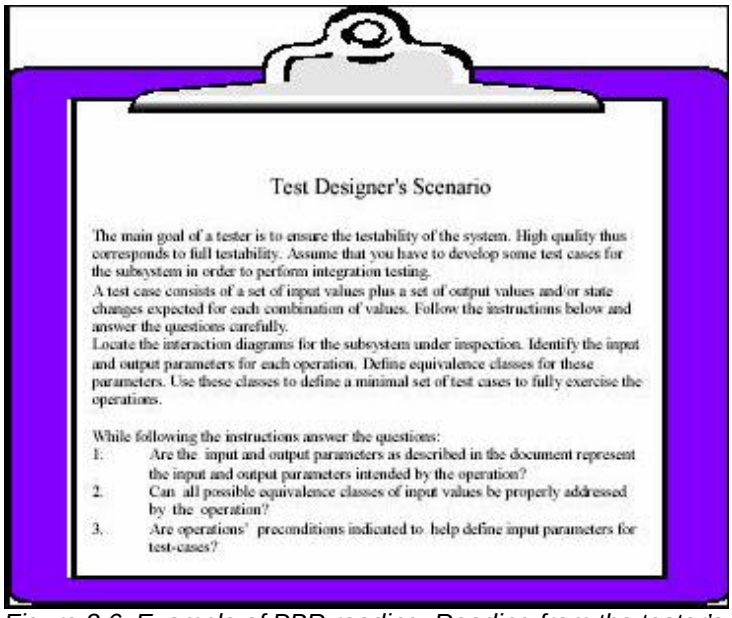


Figure 2.6: Example of PBR reading: Reading from the tester's perspective

PBR follows in general the same inspection process as the one used at Ericsson. For tailoring the reading techniques to UP (Unified Process), it uses an Architecture-Centric approach [12]. This is the approach we suggest, to be used if OORT are to be tailored to RUP at Ericsson. This is discussed later in this Thesis.

3 State of the practice at Ericsson, Grimstad

3.1 State of the art

In our study of object oriented inspections at Ericsson. We have gathered data and attended an inspection meeting at Ericsson, and analyzed the process. We have covered the inspection process they use today and written about the meeting we attended with our views and observations. Data from old inspections have been gathered and presented. And we have written about experimentation and some aspects we will consider in the experiment at Ericsson.

The inspection procedures that have been adopted by Ericsson are originally developed by Michael Fagan at IBM and so forth developed by Tom Gilb. Fagan's Achievement was to make statistical quality and process control methods work on 'ideas on paper'. He reported this in the famous paper [19] (Fagan, 1976).

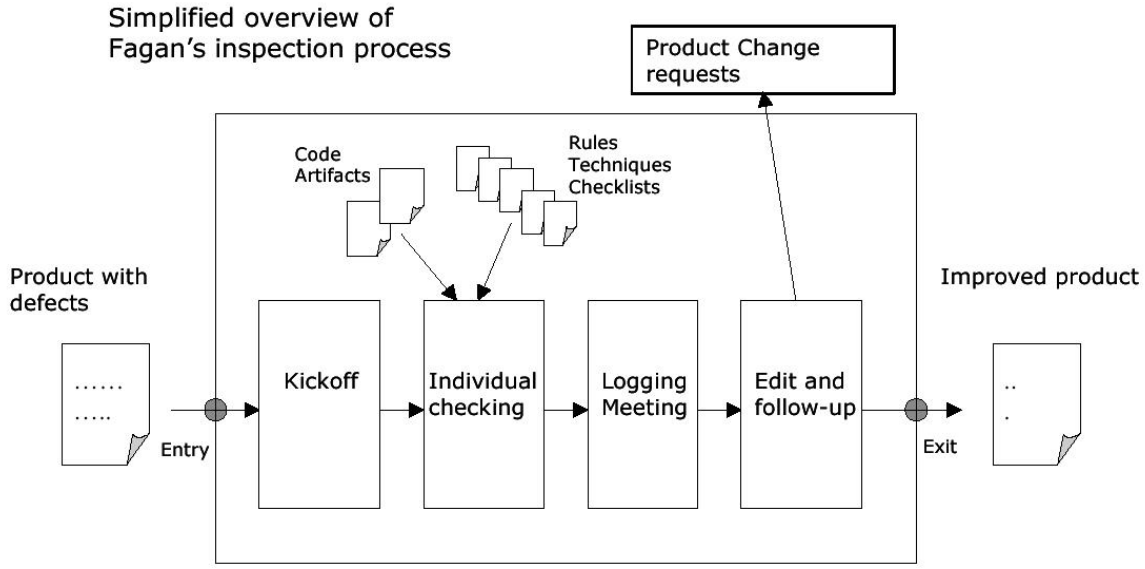


Figure 3.1 – Fagan's inspection process [13]

The inspection consists of different phases. The figure describes a single inspection cycle using Fagan's method. The stages of the inspection have different inspectors and different input and output documents.

After the kickoff, the first part of the checking is individual. Individual checking is the most important part of the inspection. Then they have a logging meeting, where they present the defects found, and they are categorized and logged. Then the log is handed over to someone in charge of resolving the defects. Then the inspector leader checks that all the defects are resolved. When the document has passed all the different stages, it exits the inspection cycle, and is ready for the next phase.

Fagan's original intention was to develop a method to reduce the number of defects in produced software. He also discovered that the method would also cut cost, development

time, support and bug fix resources and generally produce software of better quality [13]. Several large companies started using this method almost right away and there have been several enhancements of the process. Finally the techniques have been changed to meet the needs of Ericsson.

3.2 Software Development at Ericsson

3.2.1 Inspection routines

When Ericsson talks about reviews in general, they mean both peer-reviews and inspection. In the following we will talk about reviews in general, and state when necessary if it is a Peer-Review or an Inspection. Ericsson in Grimstad has adopted some of the inspection guidelines from Sweden (described in Review Guidelines, Appendix B.III), but for the most they have their own approach. This Guidelines does anyway describe certain factors that are not present in textual form in Grimstad, so we will have to draw some aspects from this Guideline and comment whether this is the practice for Grimstad or not. If Ericsson wants to strengthen their current R & I method besides what is proposed in this thesis, it is natural for them to look in these guidelines.

A review is done after the UC analysis. The designer has made diagrams and classes of high level abstraction, and a review is performed to see that the designer has understood the Use Case. This is typically called an analysis phase. After that comes a Design Phase. In RUP it is referred to as Identify Design Elements and Design Components. It results in diagrams and design elements. A review is performed after this to see if the designer has done a proper design.

Ericsson has either UC/scenario scope of a review or they have architectural-centric (subsystem) scope. A sub-system scope could include several UC's. UC/Scenario scope is used in UC realization reviews. They are easy to plan (iteration plan), but could be more difficult if the Use Case/scenario affects more than one part of the system. The drawback is that this approach will focus on the same things as the Test will. And it could most likely cause repeated reviews of the same artifact, which could lead to inspector fatigue.

Architecture-centric scope will then focus on sub-systems, processes, classes etc. Problems are that the artifact should be complete, but still they want reviews to be performed as early as possible. This can be done if only a few activities will modify the artifact. This approach is used in component reviews. This is coordinated with subsystem responsible and CM. Ericsson Grimstad are inspecting these artifacts but not in the sense of being "component review" or "architecture-centric scope" per se. They also organize it in a different way.

Reviews are planned in the iteration plan. It also happens that the same artifact is inspected several times. Reviews are performed for every RUP activity according to the iteration plan. Several activities could be included in each review. Formal inspections are

performed at the end of the iteration. In the iteration 1/3 Presentation, Walkthrough and peer-reviews are done. These are less formal then the inspection. Reviews come closest to inspections because it has similar individual preparation and meeting. Often these mid-iteration activities are postponed or not done at all because of time-pressure. Time-pressure also often causes the formal inspection to be performed half-way so that it is more similar to peer-review.

In addition the Model Structure Review is performed very early, and it gives answer to some questions that are valuable input for when and how to perform inspections/reviews. One of them is how increments will impact the model, i.e. if any parts will be updated in every increment. This is valuable if they document it and use it in planning of inspections/reviews. It is also considered how the teams will work on the model, and this is also valuable to those who should attend to inspections/reviews.

There is also Milestone reviews: Objective Review (Inception phase), Architecture Review (Elaboration phase), Operational Capability Review (Construction phase), Release Review (Transition phase). Ericsson wants the model to be conceptually whole and not changed in later increments before a review of this kind is performed. These terms are not used in Ericsson in Grimstad, but they do many of these things in their current R&I method (Milestone reviews are mentioned in Laitenberger's writings as a good companion to PBR inspections [12]).

According version control and configuration management, Ericsson's control systems like ClearCase, allows only one person at a given time to modify the model. This is why they make comments on print-outs and take them to the logging meeting. It is also crucial in their process that the models that are intended for the inspection is labeled so that all participants do their preparations towards the same version. It is labeled in ClearCase by the Author. After the review it is labeled "reviewed" in ReqPro. It has been discussed at Ericsson in Grimstad how this could be done.

3.2.2 Development Process

A product is realized by several projects. A project goes through all phases of the RUP lifecycle, Inception, Elaboration, Construction and Transition. A project produces a product release. A project can start before the previous project has ended. A project consists of several iterations. An iteration consists of all the core workflows e.g., Requirement, Analysis &Design, Implementation and Test. An iteration must end before the next iteration starts. An iteration results in some kind of executable program.

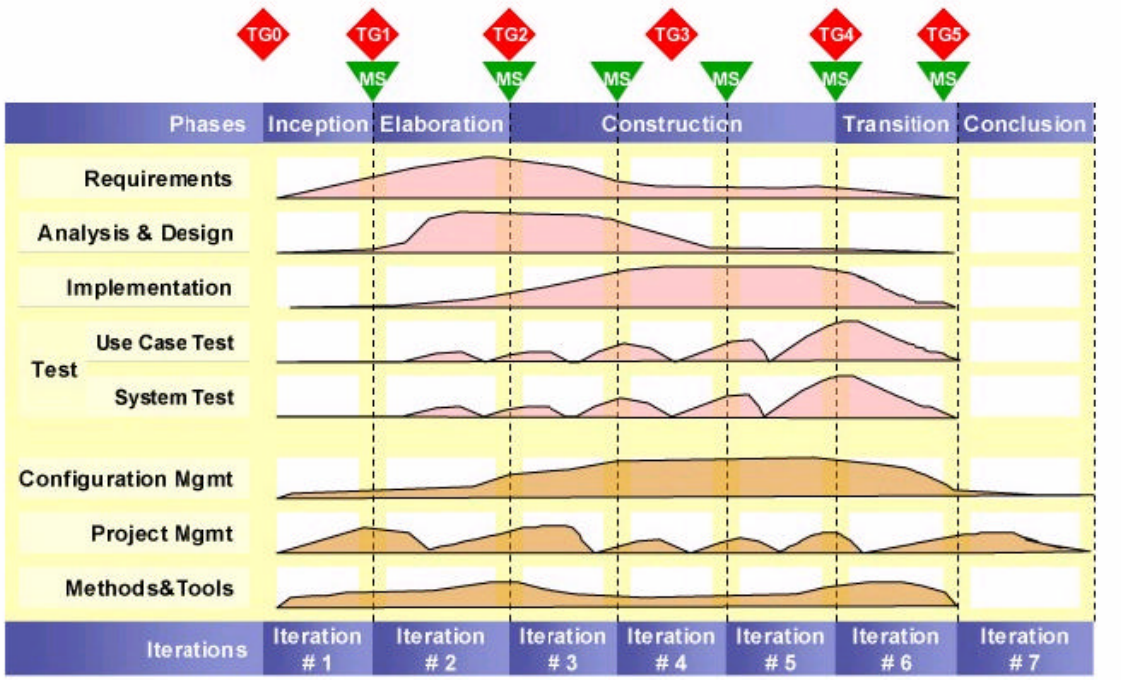


Figure 3.2 – GSN PUP adaptation

3.2.3 Requirement workflow, artifacts and structure

This workflow stretches mainly over the phases Inception, Elaboration and the first iteration in the Construction phase. After that, requirement management is mainly about handling Change Requests (CR).

Before the inception phase starts (pre TG0) the ARS (Application Requirement Specification) exists in the ReqPro database. The ARS captures customer requirements, product legacy and requirements from standards. The ARS serves as a starting point from the product management to different projects in the Project management. Between TG0 and TG1 the Functional Requirements are sketched in Rose as a Use Case and is also placed in ReqPro. They capture the actions that a system *must* be able to perform. The non functional requirements are written (sketched) down in a Word document. They capture requirements as usability, reliability, performance and supportability. These three artifacts come from the product management and serves as a input to another document called the FIS. The Project management decides through the FIS what parts of the ARS that are applicable for the project. The FIS tells if the ARS can be accepted or not and tells how the Functional and non-functional requirements can be further detailed for the right A&D level before TG1. The output from this iteration will be an accepted ARS and a detailed Use Case and detailed non functional requirements.

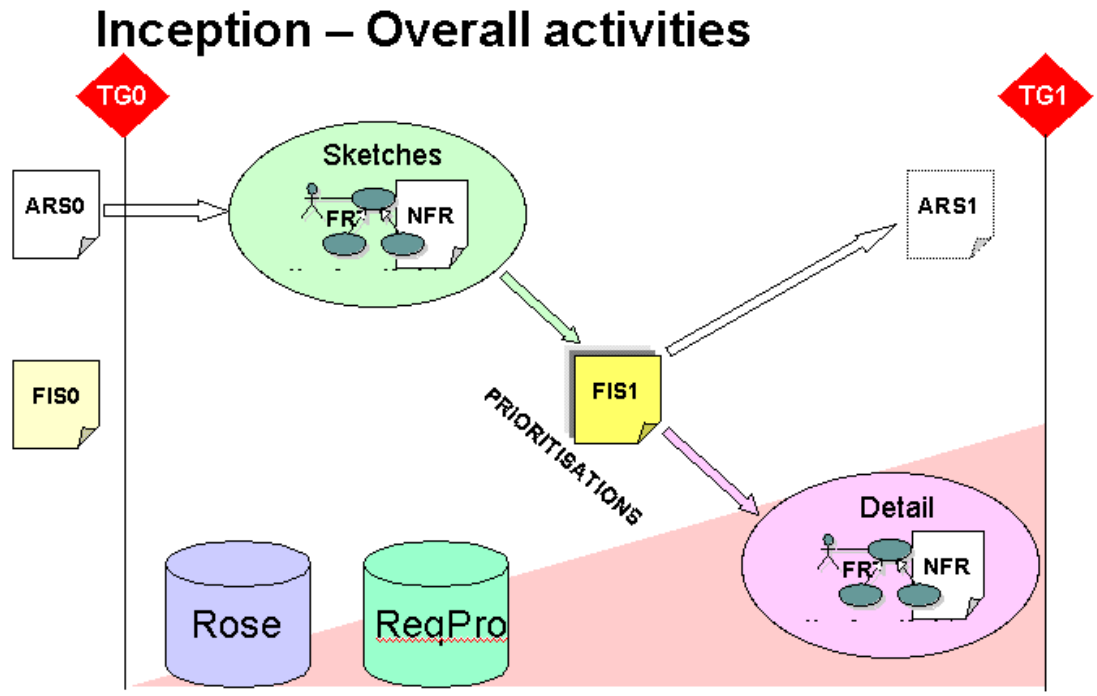


Figure 3.3 - Artifacts and activities in the Inception phase

In the elaboration the same steps will occur. FIS1 is renamed FIS2 and ARS1 is renamed ARS2. In the Construction phase the last details are laid and requirements should be final. The transmission phase mainly handles requirements in form of change requests. ARS2 and FIS2 are used as input in at least inception, elaboration and construction.

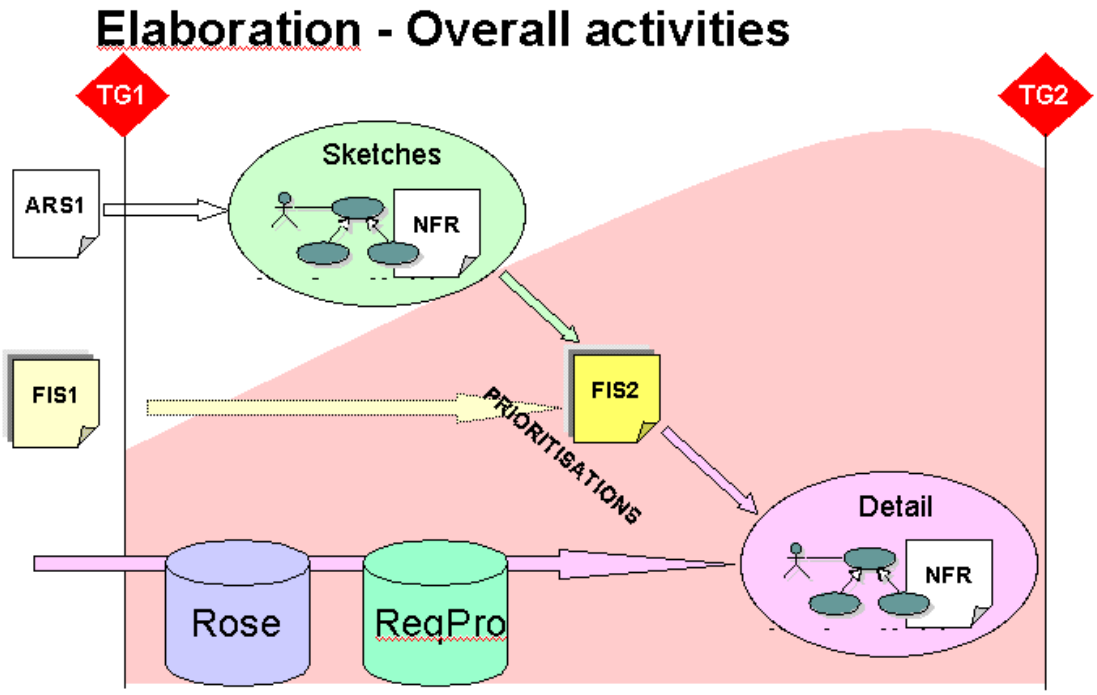


Figure 3.4 - Artifacts and activities in the Elaboration phase

The requirement Model is a model of the software requirements. It can be seen in the Meta Model and is connected through its documents and diagrams with other models of the GSN software system.

A requirement model consists of both functional and non-functional requirements. The functional requirements are described in the Use Case model and the non-functional requirements are described as normal text in Supplementary Specifications. The product will be developed over time in several projects, which means that one project will generate one or more product releases. This implies that there are both product and project requirements. The product requirements describe all the requirements on the product while the project requirements only describes the requirements that will be developed and implemented in a specific product release. Both product and project requirements are captured in a requirement model.

Figure 3.5 shows the requirements types that exist as database items in the ReqPro tool. ReqPro ties the requirements together. It is possible to print them out on paper and some but not all also exists as documents in addition to being represented in ReqPro. Refer to the figure 3.5

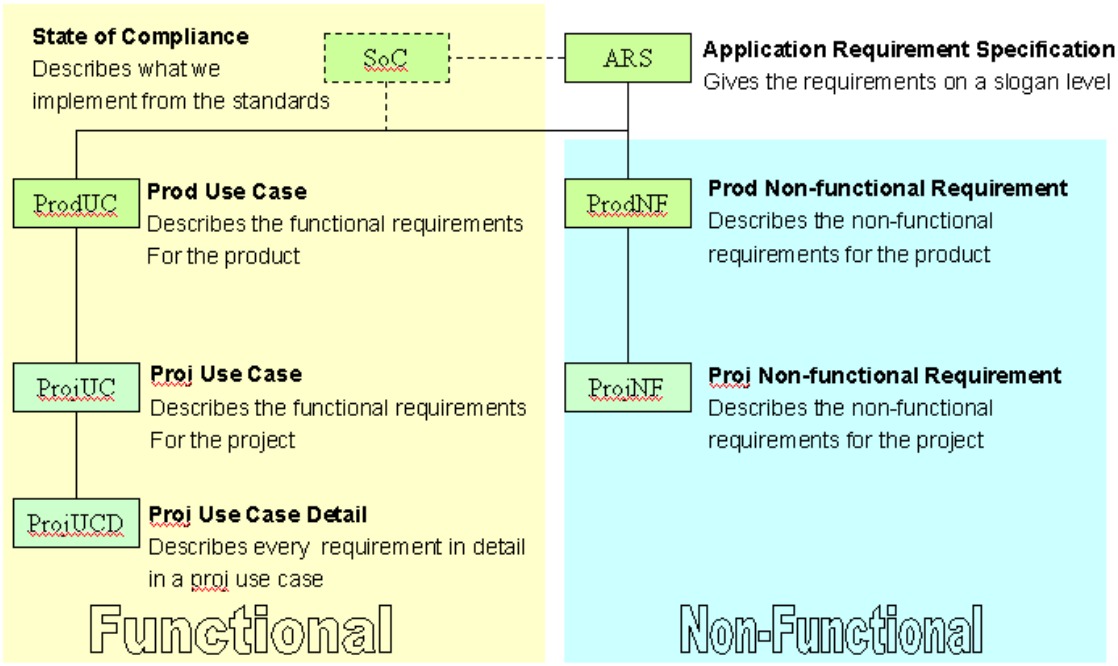


Figure 3.5 - Functional and non-functional requirements types in ReqPro

ClearCase is a tool that organizes all tools and artifacts into an environment that let one worker work on one artifact at a given time. It works like a library; you check one book out, you read use it and finally you put it back.

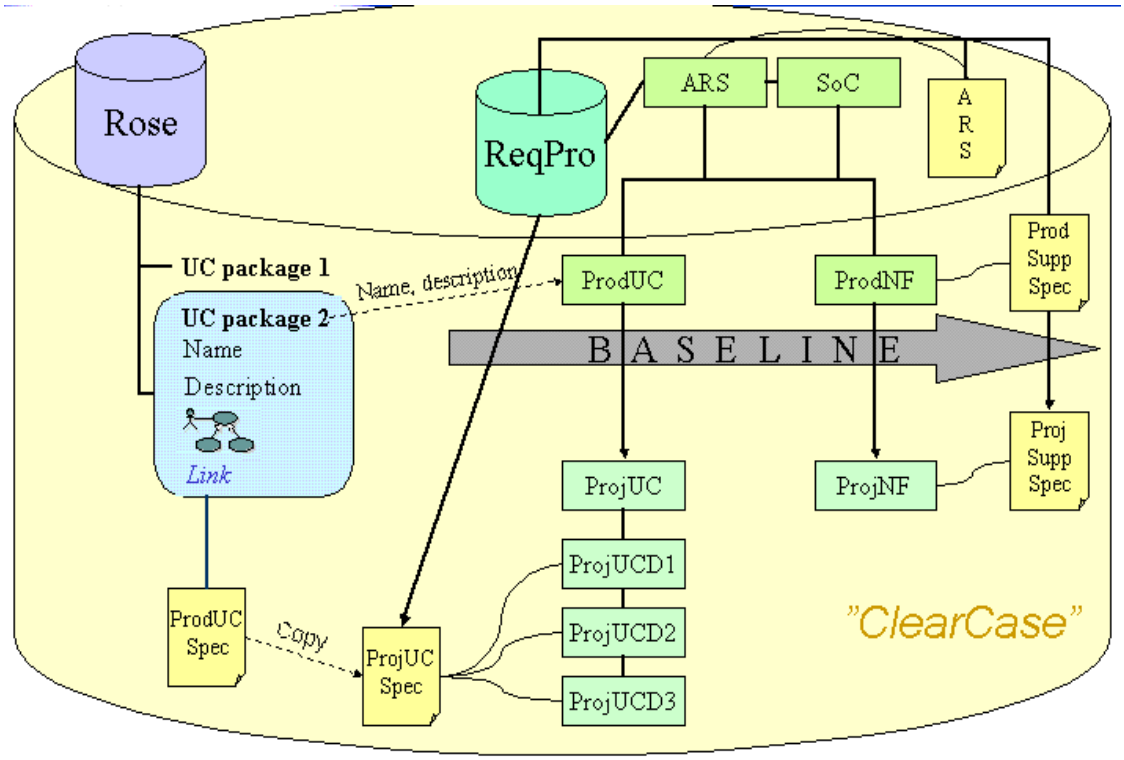


Figure 3.6 - Organization of the tools and artifacts in ClearCase

The baseline is updated through each iteration. Soda, a tool that generates reports from all Rational tools, is not used at Ericsson, Grimstad. There are also three artifacts not present here. That is CR, FIS (Feature impact study) and the development case. We have explained FIS that is only a preliminary document, and CR is simply a Change Request, written as a request for changing the requirements. The development case is a plan for how to reach the goals set in a project. See appendix B for the requirement types and their attributes.

3.2.4 Analysis & Design workflow, artifacts and structure

This workflow stretches mainly over the phases Elaboration and the first two iterations in the Construction phase. The main artifacts here are Analysis Model and the Design Model. The analysis model is created in the Elaboration phase, and is updated in the Construction phase as the structure of the model is updated. The software architect is responsible for this artifact.

The design model primarily sets the architecture, but is also used for analysis during the elaboration phase. It is kept consistent with the UC model and the implementation model. The software architect is responsible for this artifact, but designers are responsible for packages, classes and so on. We will explain these two models further, later in this chapter.

The activities in this workflow can be described with the following flow chart.

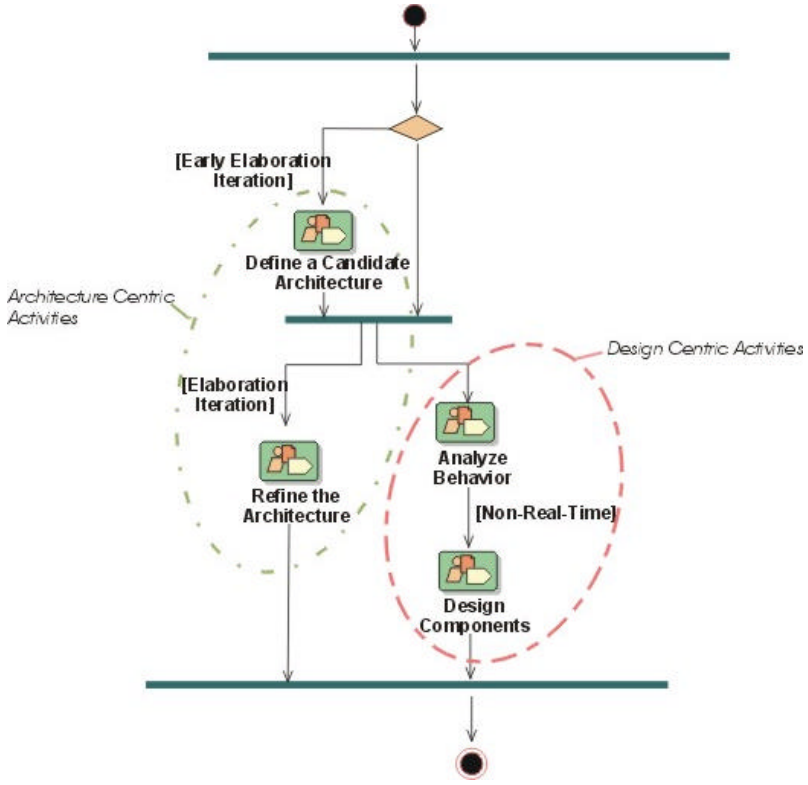


Figure 3.7 – Flow chart AD workflow

Observe the two main categories of A&D activities. As the GSN projects will not build a new system from scratch (there is an existing design base), the architecture centric activities will focus on improving and evolving the current system architecture to meet future requirements. The design centric activities, on the other hand, will focus on implementing new functionality in the existing system.

Architecture centric activities:

These activities are for the most concerned with the Analysis Model, but also interacting with the Design Model. Architects have the main responsibility. For input and output artifacts for each of these activities, see the appendix.

Define candidate architecture:

Architectural analysis:

The architect uses experience to create high level packages in the logical view (Meta Model, Appendix B.VI) in the Analysis model and Design model. It is an initial structure for the design model made out of high level components. Further on the technical solutions that are not described in the SAD, shall be documented in small Design Decisions.

Analysis of architecturally significant Use Cases:

Then the architect expresses the Use Case functionality by identifying Analysis Classes and creating Main Flow Sequence Diagram and VOPC in the Analysis Model. The identification of analysis classes is based on the flows in the UC. Documentation in the SAD, or in the Design Decisions.

Refine architecture:

Incorporate existing design elements:

The architects incorporate and refine the result from the design teams. Reuse will be incorporated where possible based on the subsystems and/or components interfaces. The result from analyzing the UC done by designers in the "analyze behavior" steps are validated and included in the analysis model as new and/or changed analysis classes. Mapping between design and analysis model is refined and SAD and Organization of the design model is updated if needed.

Review Architecture:

When architecture is refined there's always being conducted reviews. There are different concerns depending on the development status. They also detect potential mismatch between architecture and requirements. There is also some "reverse-engineering" from the actual Design Model.

Describe Distribution:
(Architects and designers together).

Defines the distribution of several processes across the physical nodes, in the system. This is done by defining architecturally significant process environment blocks and applications that will be further modeled by designers. This is documented in the deployment view of SAD.

Describe the Run-Time architecture:

Model elements are distributed among processes and process lifecycles as well as the concurrency requirements are defined. This is documented in the process view of SAD and in the Process model in Rose.

Design centric Activities:

These activities are for the most concerned with the Design Model, but also interacting with the Analysis Model. Designers have the main responsibility. For input and output artifacts for each of these activities, see the appendix.

Analyze behavior:

Use Case Analysis:

Identification and definition of new Analysis Classes and high level design elements (subsystems and blocks). New Analysis Classes means that they are not already present in the Analysis Model. The Analysis is closer to Design Elements than typical "RUP"-Analysis Classes. Next is to express UC behavior in Sequence diagrams and to create VOPC in the Design Model. All Analysis Classes have descriptions in the Design Model.

Identify and create design elements:

Analysis Classes are transformed into design elements and refining the Design Model without affecting the defined architecture. This means creating model elements that fit into the architectural structure in the Design Model. The Meta Model and the Modeling Guidelines tells what kind of elements to use. Steps include identifying and creating Subsystem Interfaces, Blocks and Block Interfaces. State diagrams connected to an interface are used when needed.

Use Case Design:

Refining, structuring and simplifying of existing UC realization diagrams in the Design Model. New are created if needed. Interaction between Design Elements is showed using Sequence Diagrams. Details are hidden in sub-sequence diagrams, linked to the diagram. Homogeneity and consistency are ensured for the Design Elements regarding names, behavior and attributes. In preparing for review, those parts of the model is labeled and linked to all documentation.

Design Components:

Review relevant Use Case Realizations:
Focus is to ensure that all requirements are met and secondarily to ensure consistency towards Modeling Guidelines, Meta Model and SAD. Following the last three mentioned ere also evaluated. Results are documented in review record. Necessary actions in case of defects are taken.

Subsystem Design:

Use Case Realizations are refined by creating Subsystem and Block Package artifacts. Interfaces are detailed and assigned to created components in the Component View.

Review Design:

This activity is performed when all diagrams are updated and complete. Focus is to review the static entities in the model and to ensure that all requirements are met and to ensure consistency towards Modeling Guidelines, Meta Model and SAD.

The analysis and design workflow work in the Logical View of RUP. The typical structure is:

Logical View
- **High level package (I.E. SGSN-GT)**
  - Analysis model
    - Use Case Realizations
    - Analysis Classes
  - Design Model
    - Use Case Realizations
    - <<Subsystem>>AAA
    - <<Subsystem>>BBB
- **High level package (I.E. Business Specific)**

And so on (look at the Meta Model in Appendix B.VI).

The analysis model is supposed to be a bridge between the UCD and UCR. The analysis model shall describe architecturally-significant use case realizations in terms of analysis classes. Analysis classes are grouped in each HLP as a package called "Analysis classes". Analysis classes have a description explaining its purpose and responsibility. Besides Analysis Classes there are Use Case Realizations.
Use Case Realizations include:

- Class diagram, "VOPC – (view of participating classes)", showing the analysis classes participating in the Use-Case Realization. It shows the design elements that are needed for that realization and shows the static behavior.

- Sequence diagrams, "Main Flow", showing how the Use Case is realized in terms of collaboration between analysis classes.

- Sequence diagrams, "Alternative flow".

The sequence diagrams describe the dynamic behavior.

Design model

The design model describes all use case realizations in terms of design objects such as subsystems, blocks, units and modules. It is organizes in "Design Packages". These design subjects has a double meaning as they are both design model entities with RUP terminology and also implementation model entities as products in the product structure. Only modules (Erlang, C or java files) realizing interfaces shall be shown.
State charts are used in the design model to describe objects with a Finite State Machine (FSM). State charts are not made per Use Case Realization, but belongs to the "object". Modeled objects are put in <<unit>>.

Design elements are grouped together in the subsystems like this:

Subsystem -> Block -> Unit -> Module

Subsystems, blocks and sometimes units have interfaces and data types. These are specified in the Rose-model so that it can be generated IDL-code from them, using IDL add-in to ROSE (Later on it will be CORBA add-in). See appendix B.IV for details on what the subsystems include.

3.3 Adoption of the techniques at Ericsson, Grimstad

3.3.1 The Inspection Process

The inspection team is made up of a moderator (the leader of the process), inspectors, authors and eventually secretary. They all contribute to the inspection process and have certain responsibilities [14].

1. Planning
2. Overview meeting (optional)
3. Preparation
4. Inspection Meeting
5. Discussion Meeting (optional)
6. Casual Analysis (optional)
7. Rework
8. Follow-Up

1. Planning

The moderator will evaluate the quality of the documents to be inspected, but mainly it is the Authors task to assure this. It will also be checked whether all the applicable input documents have been used and whether these have been inspected or used (Documents used by the Author during design will be marked). If there are big amounts of documents, they should be chunked into applicable parts so that the inspection meeting will not last for more than two hours. This is due to human tiredness. It should be considered to only inspect samples if there is lack of time. The result from this inspection can be used to decide if it is necessary with a full inspection.

The moderator will decide upon team size. Typical team size will be 3-6 persons due to viewpoints necessary, defect tendencies and volume of input documents.
Inspectors will be assigned to different viewpoints in cooperation with the moderator. The viewpoints will be assigned with regards to talents and interests of the Inspectors. Finally the Moderator sends out an invitation to the participants, with specification of the task that is to be performed.

2. Overview Meeting

This is optional, but recommended if the process or the inspected item needs to be introduced. This is an informal meeting, guided by the moderator.

3. Preparation

This is the phase where the Inspectors do their individual preparation and defect finding. Due to recommendations in the invitation and eventually at the overview meeting, the Inspectors will plan their preparation. The documents will be studied by the Inspectors, focusing on applicable parts for the viewpoint assigned. An inspector uses checklists to identify defects in the inspected item. The checklists follow this report in the appendix. Defects or issues found that not relate to the viewpoint, should also be noted. The issues are written down and categorized directly in the document. They use a red pen for major issues and a green pen for minor issues. Only major issues are reported, Minor Issues are only handed over. The result (hours used, pages inspected and defects found) are notified the moderator.

These following suggested viewpoints are meant to be a help to get started, but there might be other helpful viewpoints. They are only used as a guide to remember the most important view points.

View point	**Description**
High level	Is the artifact consistent with the high level documents (standards, requirements...)
Design Rules	Have applicable design rules been followed?
Modeling guideline	Have applicable modeling guidelines been followed?
Programming guideline	Have applicable programming guidelines been followed?
Superfluous information	Check for unnecessary information that can be skipped
Language	Is the language in the document appropriate?
Customer understanding	Does the customer understand the document?
Usability	Is the level of the information relevant for the user? Will he/she understand?
Testability	Inspect from a testability point of view
Maintainability	Inspect from a maintenance point of view
Interface	Check all interfaces, are all interfaces described?
Characteristics	Are the system characteristics sufficient?

Table 3.1 – Suggested view points

At the end of the Preparation, the Moderator has to decide upon whether the result of the Preparation is good enough to continue with an Inspection meeting. The inspection meeting will be postponed if there are too many defects and/or the Inspectors have made an improper job.

4. Inspection Meeting

The main goal with this phase is to find more defects, not only report those found in the preparation phase. There is a strong emphasis on guarding everyone's ego. The focus is cooperation, not defense and attack. The focus is on documents not persons.
Long discussions will be stopped and assigned to the optional discussion meeting.
There is a focus on serious defects. As an Inspector you are not supposed to dump out all the defects you have found.

The Moderator decides upon the exit criteria for the meeting. As a rule of thumb, the exit criteria for the inspection meeting are not met if; the rate at the meeting has been higher than twice the recommended figure or the number of defects found per page are more than three times the average of that document.

Finally authors will be assigned to do rework. Time is assigned to the task.

5. Discussion meeting

This meeting is optional and can also be performed after the optional Casual Analysis. Only relevant people to the discussion participate. Items marked with "Dis" in the defect log will be discussed. Moderator updates the defect log if necessary.

6. Casual Analysis

This is optional because it often is performed as a separate activity. Everybody that attended the inspection meeting is participating. Typical activity is to select one (or a few) of the defects found for further analysis. Then they will try to find the defect generator and give proposals on how to remove it. The findings are written down in a CAR (Casual Analysis Report) by the Moderator.

7. Rework

The author puts action codes on the issues. These are "C" for Corrected, "R" for Rejected and "N" for Noted. The author notifies the Moderator. Inspection surveys are updated with actions taken and time used. CR (Change Request) is written by the Moderator for defects found in CM controlled input documents.

8. Follow-up

The Moderator will now review the updated document. He or she uses the action codes on the defect log as a help in this process. Then the moderator evaluates the exit criteria. CR's are distributed. Inspection Surveys and Defect Log's are stored.

3.3.2 Classification of defects

The only classifications of defects are Major and Minor. Major defects are typically a sentence/paragraph that the author later may have to explain/clarify. I.E the project will save time and effort by rewriting the sentence at an early stage.
Minor defects will cost the same to fix, whenever in the project process it is done. I.E there will be no doubt of the exact meaning, even if the minor defect is not dealt with.

3.3.3 Inspection process guidelines

There is some support material available for the inspection process. These are Modeling Guidelines, Checklists and Lazy Dogs. See appendix B.V.

3.4 Our experiences with the process

We attended an inspection meeting at Ericsson on February 6th, to compare theory and practice. We have only attended one meeting so we can't speak for all meetings, but we got a general idea. The inspection was, according the participants a good example on how the meetings used to be. Our experiences with the process are described in this chapter.

3.4.1 Inspection meeting

Attendees: 4 (plus two spectators)
Participants have prepared themselves in different areas (roles)
Preparation time: About 1 hour pr. participant

The reason for this inspection was; a new diagram was "finished".

The UCR diagrams were new and old sequence-diagrams, described as;

- Sequence diagram, Analysis Model (SGSN-GT).
  Level (UCR per.): Use Case; Handle MS mobility
  Main flow: Inter SGSN Routing Area update, new SGSN

- Sequence diagram, Analysis Model (SGSN-GT).
  Level (UCR per.): Use Case; Handle MS mobility
  Main flow: Inter SGSN Routing Area update, Old SGSN

  ClearCase View: Metro_architect

Ordinary approach in practice at inspections:

- Walkthrough of diagrams (Often no time for this)
- Investigation of diagrams
- Final Investigation of diagrams

This inspection was a combination of the two first approaches, though it was supposed to be a final investigation, and all tree approaches are often done in one inspection meeting. At this meeting they only inspected sequence diagrams. The procedure at this meeting:

- ? Step-by-step inspection technique (start to finish trough the sequence diagrams)
- ? The participants comment all they can think of, and what they found during the preparations
- ? Discuss comments given on the diagram
- ? Discuss what the messages does and why they are done
- ? Discuss how detailed the diagrams should be

Comparison of the diagrams occurs in the preparations, done by the different participants/roles. In this inspection one of the roles had as a view to check the diagrams against UCS. Results from the meetings are only noted, often on the printout of the diagram, to later be corrected. There is not written any defect log over the results or errors.

The roles with view:

New Sequence diagram vs. Old Sequence diagram
New Sequence diagram vs. UCS (Use Case Specification)
Old Sequence diagram vs. UCS (Use Case Specification)
RD (Resource Deployment) impacts
SM (System Management) input
Check against Implementation/code
Understandable diagrams/documents

During the preparations the documents are rarely inspected horizontally. It will be inspected horizontally if one feels it to be necessary. They “click” their way to the different diagrams in the Rose model. This was done if necessary during the preparations. Vertical checking could be checking diagrams/documents in the analysis model against diagrams/documents in the design model. The Meta Model for the GSN project has these levels:

1. Requirements Model
2. Analysis Model
3. Design Model
4. Implementation Model
5. Process Model
6. Deployment Model

What diagrams/documents that belong to each of the levels are listed in the GSN modeling Guidelines.

3.4.2 Our evaluation of the meeting

This is our evaluation of the inspection meeting, and the inspection techniques.

About comments in the sequence diagram:

When the participants checked the comments, it was weighted on why they do what they do. Project specific limits should be left out of the analyze model. It has nothing to do with the architecture either. E.g. “Check that not more than 7 UDP….” “...and if it goes wrong, do this and that…” It’s enough to say that it will be checked. The diagram has to be consistent. They want to stick with words and expressions which, historically is much used at Ericsson.

About the sequence diagram:

There has to be a balance, regarding whether the sequence diagrams should be on an architectural or a functionally level. High level classes are included here but are relatively detailed with regard to functionality. In an analysis model it is not so important what elements are called, that was more important in the design model (There were some differing attitudes amongst the participants here). In the diagram there are elements from different project sites. The context is scattered, and there can be an inconsistence problem because of this.

About the process:

It was one moderator at the meeting. And there was one, later two which made notes on the documents. There were one amongst the attendees which was not prepared, but he probably knew more about the system than the rest, and he had a considerable contribution to the meeting. Some rework can lead to the fact that one has to write CR’s. E.g. it was brought to attention that they might have to make some changes in the UCS.

Some of the planned participants did not show up, but the inspection was performed since they thought that the most important roles, except from one, were covered. The inspection was meant as a final inspection but developed, because of bad preparations, to a combination of a walkthrough and inspection. They considered one more inspection after this one. Generally the participants where positive to inspections, they saw the importance of it. One said that it was important with a walkthrough first. They didn’t have that in this inspection, but it was preferably, it was only a matter of time-pressure.

Some of the documents where not included here, because they hadn’t been tested yet.

Other observations:

The inspection meeting seemed very educational, since it was people from different departments with varying experience which enlightened each other. Afterwards they would check with their superior if they where about to do major changes e.g. remove a

function. They seemed much better at modeling than one get the impression that students are. It seemed that most of the errors were found because of pure system knowledge.

The inspection was, according the participants a good example on how the meetings used to be. They meant it was much to improve regarding the individual preparations. Usually most errors should be found during preparations and not at the meeting [15].

We feel regarding the experiment, that when it is hard to get people to come to the meetings they are invited to, it may not be easy to make them do something they are not directly working with. We had to make sure that they will attend the experiment, and still that they are representative for the participants that use to come to the inspections. In our case this was solved having the experiment in one of their planned inspections. But coupling the experiment with a deadline wasn't a good choice, unfortunately.

It's a minor problem that they don't check horizontally. At least formally, in relation with an inspection.

3.5 Collected inspection data

The inspection data we have gathered at Ericsson will be presented below. It was a bit difficult to analyze the data since e.g. the inspection time and the meeting time is written in one field. And it was a bit difficult to separate the projects from each other, when they where presented in the same document.

We have focused our surveys on the GSN project, which is the one we will be working on. In appendix B.I is an overview of the inspection data from the GSN project from June 13th 2001 to March 5th 2002. It is not a good overview since some of the log hours include both preparation and meeting. Optimally they should have been logged separately. Design and code aren't separated either, but we can get a general idea about how long each process takes, so this is a part of what we will base our analysis on.

In summary of the collected data, we can see that in general at Ericsson, they don't use much time in preparations, but that is probably because it is a bit difficult to analyze models alone. So they might find it is easier to work together with that in the meetings.

We created some statistics of the data collected, and the results are also presented in Appendix B.I.

3.6 About baselining

When we started baselining at Ericsson, early in the project process, we started by reading a lot of material about Object Oriented Reading Techniques. And when we got access to the GSN project files at Ericsson, we started gathering information. It was a bit confusing at first because there was so much information, that we where flooded. But it was possible to systemize it, and get a good idea of the system.

There was little statistical data from previous investigations, so it was a problem to baseline the work at Ericsson. Since the data is a bit insufficient it will be difficult to compare the data with the upcoming experiment at Ericsson, but the total hours of preparation and inspection is possible compare with the results. This again is difficult because we do not know if it is from inspecting the code or the design and we do not know if the data is complete (some might be missing). The fact that the design and code is not separated in the statistics, which makes it even more difficult to compare data with the experiment, since we will only inspect design documents.

We were told that the defect logs are also insufficient so it would not be any use in them either. We will get a general idea of the result after analysis of the result and talking to the participants of the experiment.

4 Experiment preparations overview

4.1 Experiment threats

The result of the experiment has to be valid not only for the population it is drawn for, but also a bigger population. There are several threats to why this should happen.
There are four threats to be concerned about. That is Conclusion Validity, Internal Validity, Construct Validity and External Validity. It is nearly impossible to have this or any experiment without having to accept some threats.

4.1.1 Conclusion Validity

There has to be a statistical relationship between treatment and the outcome, with a given significance. Things that could violate our experiments are several. Assumptions about our experiment that is broken, and not dealt with could weaken the relationship. It is also important that we do not fish for a certain result. I.e. it is tempting to find subjects for the experiment that we think would give us a positive feedback. This would not reflect "real world" though. Another thing is the quality of what we use for measure. The quality should be so high on our guidelines forms and so on that the impact on the result should be minimal. We also have to consider, when designing the experiment, if the experiment would have the same outcome twice. If the power of the test is too low, then of course it weakens the relationship.

The treatments should be as standard as possible. This is because it should be possible to compare the results with other experiments. If the treatments are too different, then the results can not be compared to each other. In our case it is most important to compare with industrial experiments. Since there only have been one experiment, COPPE TR 01 [8], which is not so much to compare with. In that publication there is also little information about how the experiment was conducted. It could be useful for us to know more about this. But also, we have to make the treatment in way that makes it easy to do it in a similar way in future experiments. Lastly we have to consider how homogenous our groups should be. This has significance when we shall generalize to a bigger population.

4.1.2 Internal Validity

The result of the experiment should not be a result of some unknown uncontrolled factor. There are three different types of threats to Internal Validity; Single group threats, multiple group threats and Social threats.

Single group

Important factors to consider here are maturation. Participants can get bored or tired. It is important that we explain for the participants why we are doing so and so, and why it is beneficial. Mortality is another factor. If someone leaves the experiment, how will that affect the experiment? Can we replace him/her? Also there can be ambiguity about the direction of causal influence. Do we know what causes what? This should be considered when designing the experiment.

Multiple groups

If we have two groups that are compared to each other in the experiment, we must be aware of the differences in behavior. This is maybe impossible to avoid, but how will it affect the results?

Social

If there are two groups there is a chance for compensatory rivalry. In our case the group with the old method could try to rival with the group with the new method and try harder than they would in “real-life”. The opposite of this could be that the “old” group could get less active because they didn’t get the chance to try the new method. In either case it is important for us to explain the importance of this being as realistic as possible, so that we can decide for the best of the company if the new method is better or not.

4.1.3 Construct Validity

The observation (treatment and outcome) has to reflect the theory (cause and effect). I.e. it is important that we do not measure on the wrong factor to observe something. There are Design Threats and Social Threats to Construct Validity.

Design

Poor theoretical preparations are a threat. The theory for the experiment must be well-defined. It is also important to measure on several artifacts. Another danger than having only one operation, is to have only one method. In our thesis there will be two methods. If it is so that system knowledge helps finding defects in either of the methods, what level of system knowledge is explaining the causes in the experiment? That is another thing to consider. If we want to have control of this threat, we might have to run a survey on the participants, finding out what are their skills etc. Again, it is important that participants have only knowledge of one treatment. If they have knowledge of both, it is hard to tell whether the results come from knowing one of the treatments or if it is from a combination of them. The focus has to be that it must be as realistic as possible. The participants must not try harder only because it is a sort of a test.

The final design threat to construct validity is that conclusions are drawn on too few results. I.e. they are only based on defects found.

Social

There is a chance that the participants could act out of what they think is the purpose with the experiment. This is called Hypothesis guessing. Also, the participants could try harder, because they feel that they are under some kind of evaluation. One last thing is that they could have wrong expectations about the experiment, which could influence their performance during the experiment. Much of the social threats can be avoided if we clearly state, under the teaching of the subjects, what the purpose of the experiment are and how it shall succeed.

4.1.4 External Validity

The clue about external validity is that we should be able to generalize the results. There are mainly tree threats; wrong participants, wrong environment and wrong timing.

It is important for an industrial experiment that not only participants are from industry, but also that the artifacts and the tools are.

Wrong timing could be that the participant has knowledge about some historical elements that could influence his or her performance under the experiment.

4.1.5 Prioritize

Sometimes an increase in focus on one threat could decrease focus on one of the other threats. Because of this we might have to prioritize between the threats. The book (Experimentation in software engineering) talks about two different experiments in this context. That is Applied Research and theory testing. In our case we think it is applied research. The book suggests this prioritizing, starting with highest importance:

1. Internal Validity
2. External Validity
3. Construct Validity
4. Conclusion Validity

4.1.6 COPPE TR 01

The report "COPPE 01" deals with the only approach to use OORT in an industrial setting. The data collected from it is a little limited, since the enterprise decided to keep most of the data for themselves. There are two things in this report we might have to take account for. They address the need for an expert moderator that can answer questions and consolidate. They also experienced the danger for having lots of false positives. These two things connect with each other, since they found out that the moderator took all the false positives away. They describe "false positives" as "...data that are only apparently wrong. They alter something that is not wrong, generating a defect".

Another interesting thing in "COPPE 01" [8] is that the maximum time used on the experiment process is 8 hours. They also conclude with that Inspection and UML knowledge has considerable impact on the efficiency. This contradicts to the report which summarizes many of the academically experiments so far (ESEC 01) [20], where they claim that it has no effect. This indicates that the expertise in UML and inspection techniques is much higher in industry than in academy. But we must remember that this also can be due to familiarity with the docs and with the setting.

4.2 Experiment Design

4.2.1 Goals

There are certain goals because of the industrial setting. Reidar Conradi (OORT's: General and Technical Aspects v.1.4) [16] has stated three of them:

- Are time/effort requirements realistic in an industrial setting?
- Do OORT address industrial developing needs?
- Are there "value added" also for experienced software engineers?

4.2.2 State Variable

The state variables will be the new and the old inspection method.

4.2.3 Objects

Objects are in our case Design artifacts and some documents (Use Case Specification). The Design is Sequence Diagrams, State Diagrams etc. This is described in chapter 5. It should be enough artifacts to evaluate every method of interest. According to this the sample of artifacts used in Ericsson experiment should be covering every OORT. That means that using artifacts used in inspections at Ericsson are too few to cover the reading techniques, if not we could use samples from several Ericsson inspections.

4.2.4 Context

The experiment will be performed on-line, which means that the experiment is performed in its real environment, with real objects and subjects (participants in experiment). Participants will be professionals at Ericsson, Grimstad. The problem area is real problems, though the selection of artifacts may not be realistic.
The context as so are then specific for big industry (in Norwegian context) that develop software in a object oriented software process that also have adopted inspection routines on all artifacts.

4.2.5 Hypothesis

Based on reports we have read and the book "experimentation in software engineering" [5], we have written down some Hypothesis.

Null
H0a: The New Inspection method will find more defects than the old one.

(For H0 it must be said that more defects will not be per time unit, but defects found during performance of the technique no matter how much time it uses).

Hypothesis: Domain knowledge does not impact the inspection (ESEC01) [20].

(This hypothesis was unfortunately not tested. Instead a hypothesis according if development experience influenced the results was defined).

H0b: Development experience did not impact the inspection.

(Development experience is throughout defined in the questionnaire in appendix E)

H0c: Have been participant in former Requirement Inspections did not impact the inspection.

(Experience with software inspections is one of the questions in questionnaire in appendix E)

Alternative
Ha1: The new Inspection method will find less or the same number of defects as the old one.

Hb1: Development experience does impact the inspection.

Hc1: Have been participant in former Requirement Inspections had impact on the inspection.

4.2.6 Dependent Variable

This variable could be total amount of defects. This is not known in our experiment.

4.2.7 Independent Variables

The Inspection method, experience of the participants, quality of the design, etc

4.2.8 Subjects

The subjects are not only the participants in the experiment, but also subjects that the result shall be generalized to. Such subjects are support groups, project leader and anyone that are involved inspection of such artifacts as described in this thesis. Again this will make the results useful for other software engineering companies that use inspection of design artifacts. Other subjects are research groups, such as ESE [23] and ISERN [24].

4.2.9 Design Principle

We will have to randomize the participants (subjects). But we will follow the old routines that were to pick persons from certain division for certain views. But inside the division it should be a random pick. It is dangerous to pick persons that we think will support the experiment or that are especially interested in inspections, because this would not represent “real world”.

Since the artifacts here are typically known by the participants in the old method, we shouldn't do it otherwise in the new method (randomize artifacts).

The standard design type we will follow are "One factor with two treatments". The factor will be the inspection process, and the treatments will be the new and the old inspection methods. We will have a randomized design, since the same subject should not perform both treatments on the same object.

4.2.10 Experiment Instruments

Objects, in this case software design artifacts. Another instrument is the guidelines for the experiment and also several instruments for measurement. The latter could be defect forms, inspection forms, discrepancy reports etc.

4.3 Adapting the reading techniques

4.3.1 Adjusting to the experiment

After we studied the baseline of the inspection routines at Ericsson, the NTNU students had to use this information together with information about their artifacts and defect log routines. It was compared how they do it at Ericsson today, and how the OORT guidelines have been developed today.

In the process of performing an experiment at Ericsson, we have learned quite a lot about how it is to work in a large industrial company such as Ericsson. One of the things we had a problem with was to set a date for the experiment. When we started planning for this, we set a "working date" for the experiment in the middle of April. After consultations the date was moved to late April. And as the date was closing in, it had to be moved to May 8th, because there were no diagrams to be inspected before that. Unfortunately the diagrams to be inspected wasn't ready at that date, so it had to be postponed again to may 24th, which left us very little time to study the results of the experiment.

In order to perform a relevant experiment at Ericsson we needed to find the right inspection techniques to use, to be able to consider if there could be any improvements in the inspections at Ericsson. The students from NTNU have earlier worked with OORT, which originated from OORT teams, and developed them further in a pre diploma thesis. (The OORT teams consists of; the University of Maryland, The Fraunhofer Center and COPPE/Federal University of Rio de Janeiro). The students at NTNU have, during their thesis, preformed an experiment with students in a course at NTNU. In which they used the same artifacts they used in their pre-diploma thesis, to be able to compare the result of the techniques against each other.

4.3.2 Adjusting the techniques

To be able to perform the experiment at Ericsson, the OORT techniques, needed to be adjusted to the different diagrams they use. Since the students at NTNU had these techniques as a pre diploma thesis, they did the evaluation and adjusting of the techniques. We assisted, together with Parastoo Mohagheghi and Gunhild Lundvall in getting the diagrams to use. The diagrams to use in the experiment didn't finish until the day before, so we had a problem in getting the diagrams to use in time. But they managed to provide a set of adjusted techniques. Originally there is seven OORT's that have to be considered when performing an inspection. When they where adjusted to Ericsson there where six modified OORT's left, some with minor adjustments, which is quickly presented here.

The adjusted techniques:

- **OORT-1 Sequence Diagram x Class Diagram (VOPC)**
  - The goal of this technique is to verify that the class diagram for the system describes classes and their relationships in such a way that the behaviors specified in the sequence diagram are correctly captured.
- **OORT-2 State Diagram x Class Diagram (VOPC)**
  - Goal is to verify that the classes are defined, so that they can capture the functionality specified by the state diagram
- **OORT-3 Sequence Diagram x State Diagram**
  - Goal is to verify that every state transition for an object can be achieved by the messages sent and received by that object
- **OORT-4 Class Diagram (VOPC) for internal consistency**
  - Goal is to verify that the detailed description of classes contain all the information necessary, and that the description of classes make semantic sense
- **OORT-6 Sequence Diagram x Use Case Specification**
  - Goal is to verify that the sequence diagrams describe an appropriate combination of objects and messages that capture the functionality from the use case specification
- **OORT-7 State Diagram x Use Case Specification**
  - Goal is to verify that the state diagrams describe appropriate states of objects and events that trigger state changes as described by the use case specification

From the original techniques they have replaced Requirement Description with Use Case where it was appropriate, e.g. in OORT-7. They have also removed the class description, in OORT-2 and OORT-4, because this can be generated directly from the model with the use of SoDA. This is therefore called this "Class diagram, including a textual description". Whether this is inspected on paper or be inspected on paper depends on the size of the model or the amount of paper to be read.

The Class Diagram (CD) diagram shows an overview of the classes involved (VOPC – View Of Participating Classes). It also includes the textual description of each class and

of the behaviors. This information can be retrieved either by browsing the model using Rational Rose, or by extracting the text using the tool included in Rational Rose, SoDA.

Among the Ericsson documents, the UCS will have to function as a RD because it has to reflect ARS, SoC and also includes the UC model. Therefore in OORT-6, the Use Case Description is changed with Use Case Specification. In OORT-7, Requirement Description and Use Case is changed with Use Case Specification. The Use Case Specification (UCS) is a detailed description of the use case described as a basic flow of events and if needed several alternative flows and exceptional flows. Wherever this is stated the Use Case Diagrams are also considered. The specification is not used without the actual use case diagrams.

The NTNU students have studied examples of UCS and concluded that they aren't at the level to be used in OORT-5. This technique was removed so the OORT would fit the Ericsson documents. The technique which was removed is presented here:

- **OORT-5 Class Description x Requirements description**
  - Goal is to verify that the concepts and services that are described by the functional requirements are captured by the class description

The technique was based on finding nouns as candidates for classes and attributes, verbs as candidates' functions and conditions/constraints attached to these. The UCS's that where studied where not on this level. There are also a lot of interfaces which is included in the class diagrams, and that wasn't described in the UCS.

The complete description of the adjusted techniques used at the experiment can be viewed in appendix D.

4.3.3 Adjusting defect logs

To be able to get as much information possible from the experiment, there had to be made new defect logs, different from what they use today. The reason for this is because the error log for the OORT is very different from the one they use at Ericsson, so when we had to compare these two techniques it would be easier to do so with similar error logs.

Today the defect logs are only used by the moderator at the inspection meeting. They use it to write down all the information needed by the authors, to do the rework. (Most commonly they use the diagrams inspected as defect logs). The column CRN is the status for the defect, corrected, rejected or noted. These are filled out by the moderator after the rework. The major difference on the new defect log is the classification of defects. The usual classification at Ericsson is Minor and Major. Where Minor is rarely used. But in the OORT classification there is several classifications, as described in chapter 2.2.

DEFECT LOG

For: ______________________________

Doc. nr. ______________________, Rev. ___

Doc. title ____________________________

Location (page=P line=L section=S)	M ?	By whom?	Disc	Description	C R N

Code: M = major, ? = question to the author

Correction code: C = corrected, R = rejected, N = noted

Figure 4.1 – Defect log at Ericsson

During this experiment we wanted the participants to each fill out their own defect log, so we could easier analyze the results. At Ericsson they only write the defects/comments right on the diagrams to be inspected. The fact that they have to write a defect log might create extra work. The defect logs used in the experiment is presented in figure 4.2 and figure 4.3.

Defect Log

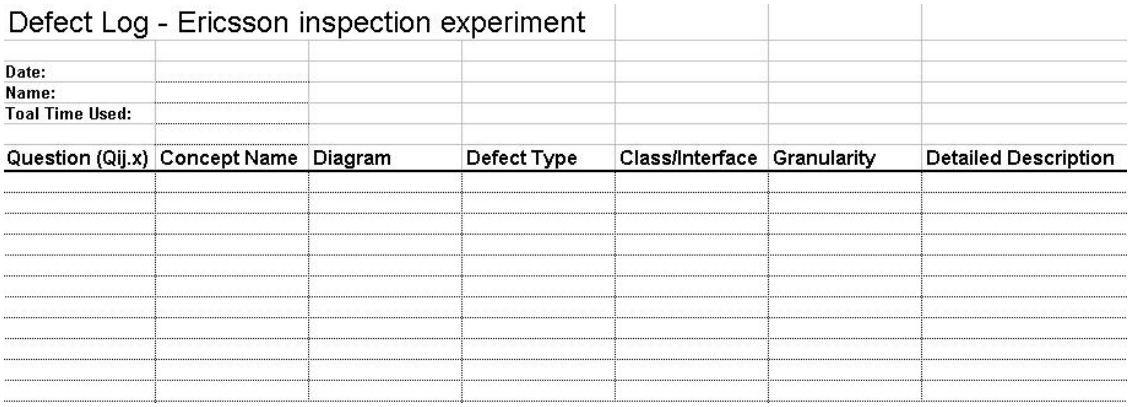
Date:
Name:
Role:
Toal Time Used:

Correction code: C = corrected, R = rejected, N = noted

Concept Name	Diagram	Defect Type	Class/Interface	Granularity	Detailed Description	CRN

Fig 4.2 – New defect log, Current R&I method

In the defect log for the Current R&I method inspection they had to write the role they had during the inspections (Which diagram they inspected).


Defect Log - Ericsson inspection experiment

Date:
Name:
Toal Time Used:

Question (Qij.x)	Concept Name	Diagram	Defect Type	Class/Interface	Granularity	Detailed Description

Figure 4.3 – New defect log, OORT

The difference in the OORT defect log is that they had to log which question they used, from the technique, to find the defect. A short description of the rest of the columns:

Concept name - This is a short term that describes the notion of the defect.
Diagram - Which diagram the defect is located on
Defect type - The name of the type of defect
Class/interface – The name of the class or the interface which is involved
Granularity - The level of detail where the defect occurs. For example attribute, behavior or relationship. Possible granularity values vary according to the diagram in question.
Detailed description - A more detailed description of the discrepancy.

Possible Granularity Values

Class Diagram (VOPC)	**Sequence Diagram**	**State Diagram**
attributes	actor	state
behavior	object	event
condition	message	behavior
inheritance	data	condition
relationship	condition	
cardinalities		
roles		
class		
interface		

Table 4.1 – Granularity values

4.4 Pre-experiment summary

The inspections at Ericsson are performed only on a few artifacts, not a whole set that would cover all the reading techniques. Another thing is that many of the defects seem to be found in the meeting.

We had to ensure that the inspection with the old method in the experiment was conducted in a realistic way, being a good measurement for the new method. This seemed to be achieved using a planned inspection. We considered using data from several inspections to compare with an experiment with the new method. The problem is that there weren't planned enough inspections to give us such a set with data. Especially, not in time to have it in April. Anyway this was postponed a little by little until late May, but we never had time to change our planning. We had to stick with two controlled experiments, performed on the same inspection.

The question then is if a comparison over a few selected artifacts is enough to draw conclusions about whether the new method is better than the old one or not. This was solved letting each participant on the new method go through all the OORT techniques and making a "fake" State Diagram, based on the Use Case Specification. These adjustments were made very hasty, because of the continuous postponing and breaking of deadlines regarding deliveries of design. This made it nearly impossible to adjust techniques to the experiment objects (documents/diagrams), and managing all the threats identified in this chapter.

The next points summarize some aspects that we feel are important to the experiment.

- The experiment should have realistic subjects. If we only had participants that were positive about new RT, UML, inspections and so on, we might not reflect "real world" in the experiment. But we must ensure that the participants will attend at the day of the experiment.

- The experiment should be given such a standard treatment that it could be approximately reused in future experiments.

- We should try our best to explain the reason *why* we are performing this experiment and *how* it should be conducted so we get valid results. This to prevent biased expectations about the experiment and the performance. Also it is motivating.

- We should not draw conclusions over too few results from the experiment. Real world are versatile.

- Participants should not have knowledge of defects, or at least if we are to compare two groups, no group should have more knowledge of this than the other.

- Watch out for false positives (COPPE TR 01) [8]

- We should consider making a survey among the participants to answer some of the goals with an industrial experiment. I.e. “Is there value added for experienced engineers?” [16]

5 Execution and results

5.1 Experiment process

The main goal of the Ericsson experiment was to verify if the OORT's are feasible for software inspections in an industrial software development environment.

It was a problem to get the experiment going at Ericsson because, it is a large corporation and it requires available time among the employees. It was decided that the SGSN-G project was to be inspected. Eventually, after a slight delay, we had three possibilities of which we made the decision to go with "Connect to PDN". The use case "Connect to PDN" was to be inspected and reworked by 15th of May. The date was further postponed, which led to the fact that we had to postpone the inspection to Thursday 23rd and Friday 24th of May.

Bjørn E. Jensen is the head of the development team of 10 people. Five of these inspected the use case and related diagrams using their old methods, and the rest used the new techniques. To be able to compare the discrepancies found during inspection, both groups need a common logging procedure, presented in chapter 4.

The decision of the composition of the two experiment groups, where made by the involved party at Ericsson. The criteria of the selection were knowledge and experience with Ericsson's inspection process, the participants modeling experience and knowledge about the current system. The groups were assigned to make them as even as possible with respect to this.

In the introduction of the experiment may 23rd, the students from NTNU presented the techniques along with a short presentation of origin of the techniques. We presented the defect logs, and how to fill them out.

The participants filled out a questionnaire about their background. This gave us the opportunity to make some judgments about experience and the usefulness of the techniques. This was a questionnaire originally used in the experiment preformed at NTNU with students, but altered to fit the experiment at Ericsson. The questionnaire was presented to get an idea of the degree of competence in the experiment. They rated their experience with 5 point scale, where;

The top two grades five and four state experience from industry. Five equals experience from multiple projects while four equals experience from one project. The lower part of the scale, two and three, express knowledge from education such as studied in class or from a book (two) or practiced in a class project (three). One equals no experience at all with the given field in software. For the complete questionnaire see Appendix E.

From the results, we see that the experience is quite equally distributed amongst the two techniques. Both groups have participants with high experience and participants with somewhat less background knowledge. Overall the table shows that participants are

professionals, hence the high representation of fours and fives. Experience with design and experience with coding, are the most influential in this experiment. When we look at the numbers from these columns the group using Ericsson's current R&I method technique might have been slightly better when it came to knowledge on coding and the technical implementation details. The design experience is more or less equal.

During the preparations of the meeting, there were not many questions about the OORT techniques. Some of the inspectors asked each other, and we got a few questions. Some of the comment and questions we got was; it was a problem getting an overview, because the CD (VOPS) was not on paper and there was many levels on the CD. It was a bit difficult with incremental systems (VOPS is incomplete, may be good for complete systems). And they had problems with inspection of parts of a system

All the diagrams inspected were in the construction phase. *Diagrams inspected in "connect to PDN":*

Class Diagram (VOPC)
Sequence Diagram
Use Case Specification
State diagram (not ready- created especially for the OORT.)

The number of techniques a developer will use regarding the OORT depends on the diagrams involved in the Use case realization. All the diagrams needed to be available on paper, as well as having the diagrams electronically available with textual comments on the computer.

During the meeting, a secretary/moderator logged the new defects. At the end of the meeting, we tried to gain some oral feedback from the developers. See more in chapter 5.4.

5.2 Results overview

From the experiment two types of results were gathered, the discrepancies from both individual reading and inspection meeting, and the time spent on each of these phases from each of the participants. We concentrated on the data from the whole inspection instead of the individual results. This was because during the baselining there was no individual data, and to compare the data of the baselining and the experiment is easier when presented as similar as possible.

Table 5.1 and 5.3 shows the data collected. First the number of discrepancies and the sum of discrepancies discovered during the whole inspection is showed. Then it shows the time spent for preparation and in the meeting. A percentage is calculated to indicate the ratio between discrepancies found in the preparation and the meeting. Efficiencies are

computed as discrepancies found per hour in table 5.2 and 5.4. For a graphical presentation of the data, see figure 5.5 and figure 5.6.

The most obvious observation made from these figures is that considerably more discrepancies were reported using the OORT's, 25 versus 39, but they also consume more time. This causes the average efficiency to be 1,37 versus 1,13, overall in the inspection.

The results of the experiment can be viewed fully in Appendix F

Current R&I method

Here we see that they found more defects during the preparations than in the meeting. Note that the participants might have spent more time on preparations now than they usually do because this was an experiment.

Total defects	**25**	**100 %**
Preparation Defects	17	68,00 %
Meeting Defects	8	32,00 %
Total hours	**18,25**	**100 %**
Hours Preparation	10	54,80 %
Hours Meeting	8,25	45,20 %

Table 5.1 – Statistics from the current R&I method inspection

Preparation	**meeting**	**Preparation + meeting**
1,70 Def./hr.	0,97 Def./hr.	1,37 Def./hr.

Table 5.2 – Defects per hour, current R&I method

OORT

Here we see that almost all the errors were found during preparations.

Total defects	**39**	**100 %**
Preparation Defects	38	97,44 %
Meeting Defects	1	2,56 %
Total hours	**34,5**	**100 %**
Hours Preparation	25,5	73,91 %
Hours Meeting	9	26,09 %

Table 5.3 – Statistics from the OORT inspection

Preparation	**meeting**	**Preparation + meeting**
1,49 Def./hr.	0,11 Def./hr.	1,13 Def./hr.

Table 5.4 – Defects per hour, OORT

Graphical comparison

When we compare the two techniques, we must also consider that the OORT was a new technique to learn, and that might have taken extra time in preparations.

Here we see a graphical presentation of the total number of defects in the two techniques, divided into preparations and meeting.

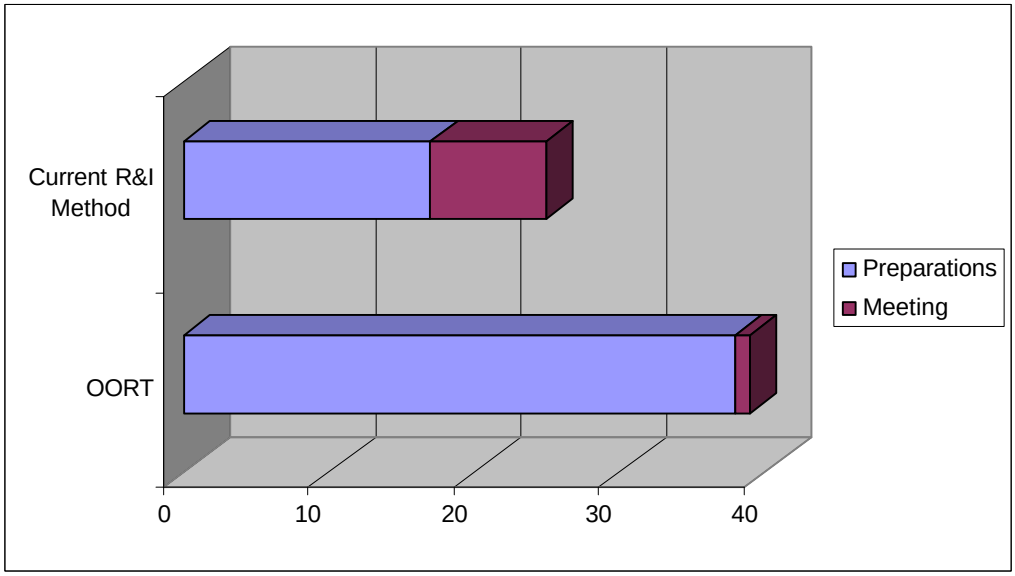


Figure 5.5 – Defects on both techniques compared

Here we see a graphical presentation of the defects per hour of the inspections in the two techniques, divided into preparations, meeting and a combination of these.

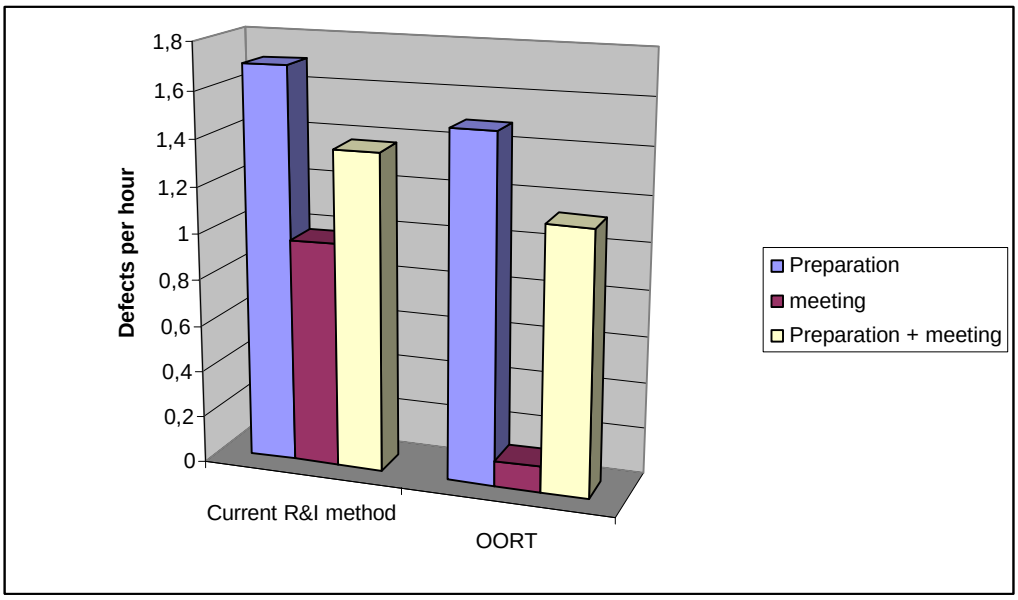


Figure 5.6 – Defects per hour on both techniques compared

5.3 Validity of results

The validity of quantitative data from an experiment such as this one is always a subject to discussion, and vital for the interpretation of the results. (Some of the general threats to experiment validity are presented in chapter 4.)

To be able to utilize all the techniques, a state diagram was extracted from the activity diagram in the use case specification. The activity diagram is a hybrid of a state diagram and a data flow chart. The team using the current R&I method did not emphasize this particular diagram. Thus, the number of discrepancies can be somewhat misleading to compare directly without considering this issue. Removing discrepancies related to state diagrams might give a better baseline for comparison. Still, techniques and diagrams can be seen as a unit. The current R&I method with diagram leads to detection of one particular set of discrepancies, while the OORT method with diagrams leads to detection of another set of discrepancies.

5.4 Comments on the experiment results

5.4.1 Comments on the current R&I method

Participants: 5 inspectors + authors + moderator (moderator was extra in this experiment (usually author)). (There was three authors present totally, but not present during the whole meeting).

Usually they have a walkthrough of the diagrams before the inspections (but there was no time for this now). During the meeting, the authors went through the diagrams step by step and the inspectors commented as the defects occurred or as they where relevant. The author of the diagram/document guides the participants through it. The typos was not included as comments, they where just sent to the authors. Participants have different focuses (viewpoints) in the inspection, and some of them just inspected one diagram. In this inspection they had their focus on UCS, SqD and the internal construction of the design. It is easier to detect technical defects when that is your focus.

The time spent on preparation in this group may have increased slightly due to the experiment setting. Participants will “prove themselves”. Normal preparation time for a regular inspection is maximum one hour per person. Preparation varied a lot amongst the participants, both in time usage and the level of which it was performed. The different group members were assigned viewpoints for their preparation. In practice this meant assigning different parts of the model (VOPC, SqD and so on) to different participants. This task was performed internally in the group, according to the usual process. We did not influence with this at all.

The result of the meeting was largely dependent on the persons. I.E when the first author left after they discussed one diagram, there was very little discussion. The participants do agree on the high value of the inspection, but time is the biggest obstacle at Ericsson.

5.4.2 Comments on the OORT method

Participants: 5 inspectors + authors + moderator (moderator was extra in this experiment (usually author)). Authors from the other meeting where present to answer questions about the design etc. (There was three authors present totally, but not present during the whole meeting).

This meeting was done in the same way as the Current R&I method meeting. They went through the diagrams step by step and comment as they occur or as they are relevant. In addition they had a state diagram which was not relevant to the diagrams to be inspected, but was added to be able to be use the OORT's. The state diagram was written from the UCS, but the inspection had a new version of the UCS, so it was not longer so relevant.

All of the diagrams where not implemented in the system before the meeting. As a consequence of this, there occurred errors which would normally be OK. (Especially that the VOPC wasn't updated). The UCS wasn't detailed enough for SqD, so it was difficult to locate names for use in the techniques

The participants felt that OORT was more systematic than the Current R&I method, and that it seemed to make inspectors more conscious with the models. The technique was good to find errors, guides one to find defects that one wouldn't have found else. It should also have been possible to check against the Ericsson standard, and should include internal rules/guides. It is also important to check against modeling guidelines for architectural construction.

It was also said that learning a new technique took away focus on going deeper looking for defects. Question then is if all the steps take away the focus on finding defects out of experience. Maybe you're satisfied when you have come through the steps and you trust blindly on them finding all the defects. A lot of the OORT seemed to be related to inconsistency between diagrams and requirements. It is good for checking the number of classes in the different diagrams.

Some OORT's should be written different to fit better at Ericsson, e.g. for CD's because CD x SqD didn't give much results for the system inspected. SqD x State gave most results and it would have been nice to implement that in the process. The OORT's are good for comparing documents in a vertical manner, as in comparing diagrams from different development cycle such as comparing the UCS with the design diagrams. The OORT's and Ericsson's current method seems to uncover different types of defects. Properly adapted and used correctly, they can uncover defects such as mismatches between the model and specifications and standards. Such defects tend to be more expensive than the trivial implementation specific ones. Ericsson might benefit from using a selected subset of the techniques. Some adaptation of them could also be beneficial.

5.4.3 Comments on both techniques

There was a big difference between the errors found in the two inspections. OORT looked at the UCS, which wasn't done during the first inspection. They look more at standards Architectural construction and the design when they are modeling. In what degree is this true? How much do they look at UCS? It was found more defects of technical value during the first inspection. IE defects regarding the order in SqD's.

The OORT may be just as good for incremental inspections, as for finished documents/models. And some of the techniques might be used and/if modified further, to fit to the Ericsson documents. But then it would be important to establish guidelines from the beginning.

There is naturally more traceability with OORT. And there are more aspects regarding the whole model during OORT inspection. The comments in the OORT meeting were more general, focusing on design and UML defects. It was said the two techniques gave a complementary inspection, which eventually would cover more defect types.

Using both techniques, implementation experience is important in uncovering some defects. The participants of the current R&I method group had a slightly more technical implementation experience and found more technical faults. One of the participants commented that if the questions were executed more thoroughly they might have led to the discovering of more technical defects. A possible change to the techniques would be to check the sequence diagram internally against some guidelines/architecture rules to see if the messages are sent in a proper order. The former group found defects of this type because they had more implementation experience.

Ericsson develops advance telecom systems based on standards in addition to requirements documents such as their Use Case Specification. A very important part of the inspection is to verify that the standards are correctly implemented. The standards are given as references in the UCS, but the OORT's do not handle this very well. Ericsson would probably benefit from a specific technique to verify standard conformation. The focus should be to verify the contents and data types of the messages sent back and forth in the system.

The OORT's are created to inspect whole systems, but they can work for an incremental process too. Ericsson develops in an incremental way, inspecting changes to the model only. In addition they develop subsets of a larger system. This takes some of wholeness and overview away from the inspection.

6 Discussion

6.1 The Experiment

We have evaluated a new set of reading techniques to be used during an inspection or a review. These techniques which consist of procedural steps guides the inspector through diagrams and help him/her find defaults by comparing diagrams/documents with each other, and asking questions that the inspector have to answer. These techniques (OORT) take longer time to perform, than their current routines. So how can we expect that inspectors will use these techniques when they are under time pressure? Even during the experiment, the participants passed several questions, to save time. The routines they use today are pretty easy, and even those are not performed as they should.

Of course, the new techniques took longer time because the participants had to learn a new technique, but design teams always change at Ericsson, so it will nearly always be someone who has to learn the techniques from the beginning. The new techniques demand more use of time from the inspectors. This could be an unwanted load for the participants. One could change the techniques a lot, making them slimmer, but would they then have their original effectiveness? And would they keep their strengths?

The comments from the experiment however showed that the OORT techniques were very useful. They found some defects that was difficult to find else, and that made the participants more conscious about the model. There was a lot of discussion concerning the model as a whole, and on the UML design in general. On the other side the old technique found more defects of technical value. This was maybe according to more implementation involved participants. But that's more hypothetical. One of the comments, that participants in the experiment seemed to agree with, was that the inspection seemed complementary when both techniques had been used. This comment, defects found and experiences from the studies, leads one to think of a possible solution where both techniques are performed. The question is *how* this could be done. This is further discussed in chapter 6.4.2.

What about the Hypothesis we defined in chapter 4?

H0a: The New Inspection method will find the more defects as the old one.

The experiment results in appendix F, show that the OORT find more defects than the current R&I method, but less effectively. It must also be said that many of the defects found in the OORT experiment was related to the State Diagram only used for that experiment. 11 defects were found that was connected to that state diagram. If they are removed the OORT found only three more defects. Tables are found in chapter 5. Therefore based experiment results that are rather sparse, we can in guarded terms say that Hypothesis H0a are true.

H0b: Development experience did not impact the inspection.

Development experience was thoroughly described in the questionnaire with measurements for development experience in requirements, design, coding, testing and other. The most interesting data here can be found in the experiment of the current R&I method. Here it is a direct connection between development experience and defects found.

Current R&I method	Development experience	Defects found
Participant 1	85	8
Participant 2	79	4
Participant 3	72	3
Participant 4	61	2
Participant 5	54	0

Table 6.1 – Hypothesis H0b, Current R&I method

This gives us a clear indication that the current R&I method relies heavily on the experience of the participants. Is this also true for the OORT experiment?

OORT	Development experience	Defects found
Participant 1	86	10
Participant 2	79	Not available
Participant 3	74	10
Participant 4	69	18
Participant 5	44	9

Table 6.2 – Hypothesis H0b, OORT

As we see there is no clear indication here. We assume these two tables show us that while the current R&I method relies much on the inspectors experience for finding defects, OORT is more dependent on how well the participants has followed the OORT's. This adds reason to the comments from participants that the current method found defects of more technical art. The OORT found other, more subtle defects that were not detected by the first look on a diagram/document.

For this experiment we can say that H0b is true for OORT. But the fact that it is not true for the current R&I method (H1b), gives us valuable indications about the nature of the two techniques.

H0c: Have been participant in former Requirement Inspections did not impact the inspection.

The participants had all good experience with inspections. All of them had either 4 or 5's on the questionnaire, except one that had "1", but he found just as many defects as the others, so we can conclude that for this experiment, inspection experience did not influence the result. In other words, H0c is true. See Appendix E.

In chapter 4, we detected some threats to the experiment. The unfortunate situation with postponing and late clarification on what artifacts to be used, led to hasty and poor preparations to the experiment. The possible threats we found in the period before the experiment gives us anyway understanding about factors that could have influenced our experiment. Following are the most actual threats. We only comment the threats that we felt was not good enough covered.

Conclusion validity

"The treatments should be as standard as possible. This is because it should be possible to compare the results with other experiments".

The OORT techniques was held as close to the original as possible. The accomplishment of the experiment however was not conducted in a way that it could be called a controlled experiment. There were too many unsure factors. The accomplishment was to a large degree result of hasty preparations.

Internal validity, Social

"If there are two groups there is a chance for compensatory rivalry".

The setting at Ericsson, with many dismissals, created some tense circumstances. We felt to some degree, maybe also because of the questionnaire that had to be filled out, that there were some tendencies from the OORT group that could have lead to increased performance. This is somewhat hypothetical. It seemed to be a general rise in performance from both groups. But since this was equal for both groups, it did hopefully not influence the result too much.

Construct validity, Design

"Again, it is important that participants have only knowledge of one treatment. If they have knowledge of both, it is hard to tell whether the results come from knowing one of the treatments or if it is from a combination of them".

The OORT group had of course knowledge of the current R&I method. This could not be avoided. The OORT group had plenty to do with coming through all the OORT's, so we can hope for that this didn't influence the result too much.

The fact that the state-diagram was only used for OORT is a crucial point that leads to difficulties comparing the two techniques in the experiment. It was possible to detect from the results what faults that were according to this diagram. That helped a little, but didn't say too much on how this influenced the time usage. It was a mistake to use this only with OORT. Again this decision was made very hasty. Actually, the state diagram was made in addition to the Ericsson artifacts, the night before the experiment.

In the experiment the requirements was based on the UCS. This is dangerous because the UCS was neither detailed enough, especially with regards to interfaces, and also this is the same requirement base that the test team uses. The latter would mean that OORT and the Test would catch many of the same defects, which is not cost-effective. On the other side the UCS was too big for the inspection experiment, not fitting into the time-limits of a normal inspection. One would have to filter the UCS to the specific item(s) that are under inspection. This is considered in chapter 7.2.

Coppe

"They also conclude with that Inspection and UML knowledge has considerable impact on the efficiency".

One of the hypotheses that should be investigated because of the results from the COPPE [8] experiment was how system knowledge influenced the experiment. This was forgotten when preparing the experiment. Further on, inception knowledge should be better defined in the questionnaire. Only one value for the inspection knowledge was too little too draw conclusions on, since the participants seemed to have an overall good experience with inspections.

6.2 Reading techniques

The PBR and the Document based reading/ Check list based reading have been throughout discussed in other papers (such as "isern-99-01") [22], and PBR has been found superior also in experiments [18]. It is difficult to label Ericsson's current R&I method. It has elements from both of the mentioned methods. The only way we can say something about the current method is through the studies and the experiment in this Thesis.

Speaking of OORT versus PBR it is clear that one thing differentiates the two reading techniques from each other. That is the principle of traceability. OORT is the only technique that compares two or three diagrams with each other consequently. Our subjective opinion on this matter is that OORT as reading technique would be superior. This of course should be investigated through experiments.

6.3 Ericsson Context

6.3.1 Workflows

The Unified Process (UP) is a generic process framework that can be specialized for a very large class of software systems. A specific instance of it is the Rational Unified Process. The RUP is component-based, which means that the software system being built is made up of software components. The distinguishing aspects of RUP are captured in three key phrases – "use-case driven", "architecture-centric", and "iterative and

incremental". Inspections use the concepts of the first and the last, but maybe not the architecture-centric concept. Doing so would give wholeness in covering defect-detection for a software development system based on RUP.

According the development process, we see that there are performed Use Case testes and System testes. Having an R&I method that based on Use Cases, one could also have, as proposed in chapter 7, an inspection method for the system, based on OORT. This is further discussed in chapter 6.4.2.

Reviews/Inspections seem to be held three times, one time during the analysis phase and twice during the design phase. This is of course not always true in reality.

6.3.2 Artifacts

Requirement guidelines wants the Project Management to be finished with 80% of the requirement detailing before construction phase starts, both functional and non-functional. So it could be beneficial to perform after that time, but we have to remember that the system are not build from scratch and also requirements are build on each other. This means that it already is a stable base of requirements available. What determines if the requirements are good enough is in what degree it describes the low level design. This has to be decided upon when picking out the artifacts for the inspection.

Designers seem to use these artifacts in their work:

- Requirements
- Modeling Guidelines, Meta Model and SAD
- Old Design

6.3.3 Structure

The structure or we might also say the architecture is separated from the design using "views". A design model can have several "views", that are different abstractions of the model, depending on who's supposed to read them.

In the experiment the VOPC from the design model was used. This was reported to be useless. The VOPC adds nothing new that is not already found in the subsystem's class diagrams. Because of this updating the VOPC are often postponed. The VOPC also gets big in size and one have to search for the actual class that are to be inspected since there are many classes involved in the VOPC that are not part of the inspection.

It is also important to have in mind when tailoring the OORT's to the model(s) that they are supposed to work horizontally and vertically. Horizontally would be inside the Design Model and Vertical would be between the Requirements Model and the Design Model.

6.3.4 Inspection

Ericsson has addressed the need to improve the routines regarding their Review and Inspection process. Especially the preparations before inspections and frequent reviews suffer from participants having time pressure. Maximum preparation per inspector in the preparations has been one hour maximum. They understand the need for inspections, but deadlines are prioritised. It is even hard sometimes to get participants to the inspections. We cannot expect the priority of deadlines to change in the future. So this problem wills most likely still be present in the future.

Same diagrams in UML/RUP are used in several models. Should we inspect models or diagrams? Test at the end of each iteration based focuses on the fact that each iteration adds new information but also changes or extends a lot of the existing diagrams and classes. By changing them it is likely that inconsistencies within classes and subsystems are introduced. Picking the right unit for inspection might be focused on a particular type of defects or on compensating for inconsistencies to previous iterations. Although at first sight this nearly document-oriented approach appears to be a good strategy, it leads to two difficulties.

First, crucial information is often distributed across various parts of a document or even across different document types. Thus, if the inspection is limited to a particular (part of a) document, an inspector may miss crucial information for a sound inspection. This is by some degree covered by Ericsson, Grimstad's use of views. Object-oriented development methods and the recent appearance of the Unified Modeling Language (UML) accentuate this problem because they usually use different types of diagrams to represent various sets of information.

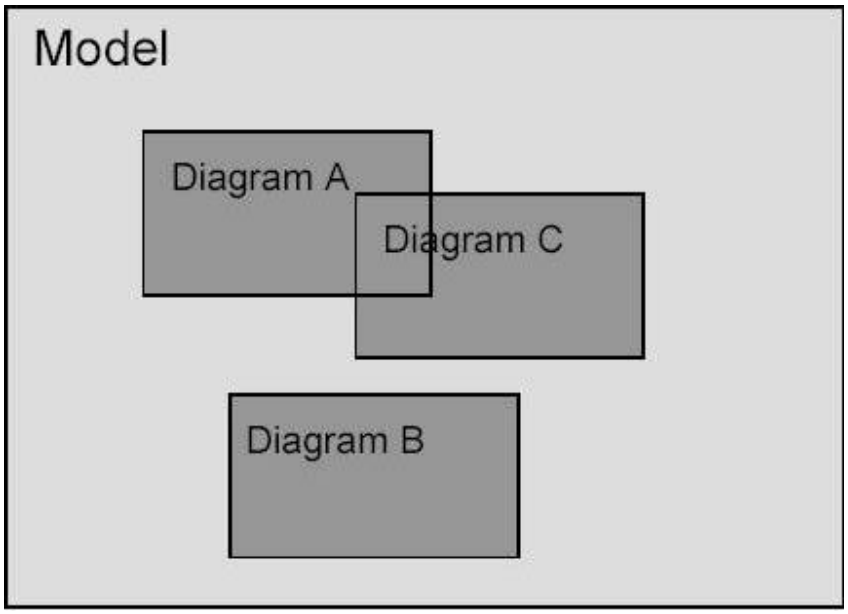


Figure 6.1 - Diagrams are "windows" onto the Model

Because of this, information about a given logical entity, such as a class or an object, can be described in many different documents, and a specific document can contain information about many different logical entities, that is, there is a many-to-many relationship between logical entities and diagrams. An inspection whose goal is to check a particular (part of a) document may end up either having to analyze many logical entities, or may only partially cover a logical entity that it describes.

In addition, basing the inspections on many off the same things would give a less orthogonal approach, finding defects of same kind. Based on cost-effectiveness it would be preferable with a more orthogonal approach.

OORT techniques seemed to be valuable in making users conscious about the whole model. This was one of the comments from the experiment. If this should give value in the A&D workflows, it should be performed early in the elaboration phase. But this might be replaced with an extensive walkthrough of the model, showing the traceability between diagrams for the design team. Walkthroughs are often postponed because of time pressure. If the walkthrough is done early, there might be less time pressure and such a walkthroughs for the team can be done. This is much according to the experience of each team. Anyway there is information about the model in the GSN modelling guidelines.

6.4 Summary Analysis

Inspections at Ericsson are for the most coupled with implementation of new functionality. Unfortunately they suffer under the time-pressure connected with the deadlines.

The experiment showed that the two reading techniques were complementary in many ways. In addition it would be positive to have an inspection that was based on other premises than the Use Case test, as is the case with their current R&I method.
The only general reading technique besides OORT that focuses on graphical notation in design is PBR. PBR ore mature in sense that it has been more used in experiments and also it has been tailored to Object-Oriented Software Development processes. OORT has a strong advantage in using the concept of traceability between diagrams/documents. Their current R&I reading technique are under constant development. It uses elements from Check-list based Reading, Perspective Based Reading and Scenario Based Reading. Using the architecture centric scope, helps organizing the use of OORT inspections since the documents/diagrams connected with an entity are conceptually whole and easier to determine for the organizer of the inspection.

At Ericsson, Grimstad they use RUP, and develop diagrams/documents incrementally and in parallel with each other. During the experiment it was said that the requirement documents was too big for the limits of the inspection. They also covered many low-level elements that were not part of the inspection. If OORT was to be used at Ericsson, a possibly solution to this could be to filter the UCS and SS through FIS documents, and in turn the UCS and FIS could filter the ARS. The references in these documents make it

possible to move in as many levels as necessary in the requirements.
The views used in RUP (Logical view, process view etc) separate the architecture from the design. Architecture in addition to old design and different requirements are the most important input for the designers. The experiment also showed that the class diagrams in the subsystems should be used instead of the VOPC for use with OORT. VOPC only shows the participating classes for all subsystems and are not always updated.

Inspections as they are now are for the most based on single documents/diagrams, but since information in UML and RUP is spread over several diagrams/documents, it should also be performed inspections over whole models. OORT fits perfectly for this, being originally made for that purpose. In addition the OORT in the experiment seemed to make participants more conscious about the model.

7 Improved inspection process and techniques

7.1 Introduction

One of the Thesis goals was to suggest improved inspections techniques and develop guidelines to be used with them. Because the experiment was performed so late, we had less time to make the guidelines. Anyway it is clearly stated in the following how these techniques (OORT) should be used in Ericsson's context and how guidelines should be further developed.

But first we will discuss the how this could be done. Either based on the same premises as the current R&I method and secondly based on a new architectural-centric approach. After the discussion we come with our solution in chapter 7.3 and 7.4. Finally we make a summary of this chapter.

7.2 Discussion

7.2.1 OORT, replacing current method

Each iteration ends with a delivery of a build. The design is therefore reviewed/inspected at the end of each iteration.

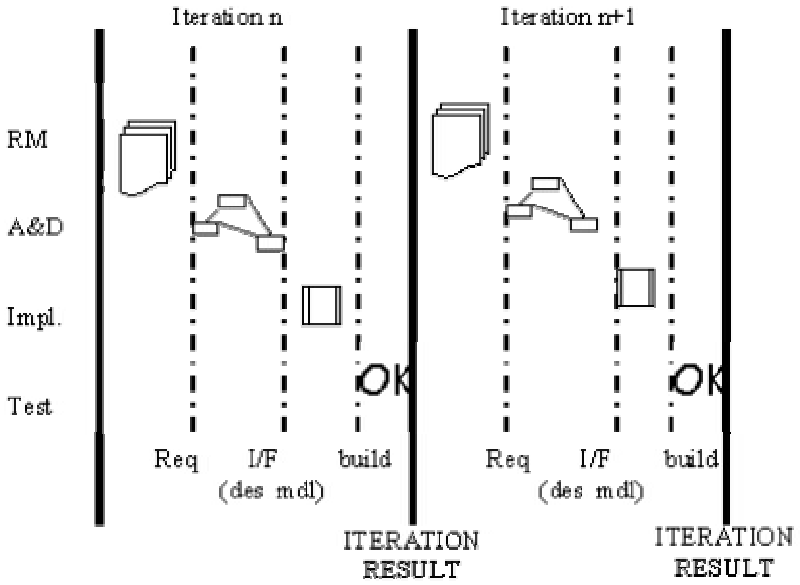


Figure 7.1 – Iterations

Let us think that a project has two iterations during the Elaboration phase. Even if it could be beneficial to perform an OORT early in the elaboration phase, since defects should be detected as early as possible out of cost concerns, it would mean that one should perform an OORT inspection at least twice during the elaboration phase. This seems very unlikely

to be executable as we saw the experiment being postponed so much as it was because of time-pressure. It is also obvious that many of the design artifacts are very incomplete until the project comes to the end of the elaboration phase. The bottom line is if not it could be best to perform OORT at the end of the elaboration phase, where all the requirements artifacts also are almost complete. It should also be considered to use OORT as a final inspection some time out in the construction phase, where many of the design diagrams are getting close to completeness. This because of the time-consuming nature of OORT inspections and the fact that "time is money". But this would be the case only if OORT are connected with the iterations, replacing the other techniques. Such approach would not be preferable since results from the experiment showed existing R&I method to be complementary with OORT.

If OORT should replace the current R&I method, the planning of inspection would be left to the Use Case team. The Use Case team sees some requirement artifacts that are tailored to what they shall do, in a ReqPro base tailored to each team. They work with their design artifacts, delivering them for build in the end of a iteration. Some of the OORT techniques require artifacts that do not directly exist inside each teams "world". This calls for the need of someone who has overview over all the artifacts involved in a bigger part of the system in the early phases of the process. That is the Inception and Elaboration phases. The question is whether its natural that sub-system responsible should take care of this, and if the UC team leader should do it, would he/she have time to do all that investigation?

Test will mainly connect their test cases to the lowest level of requirements, ProjUCD, ProjNFD and SoC's.

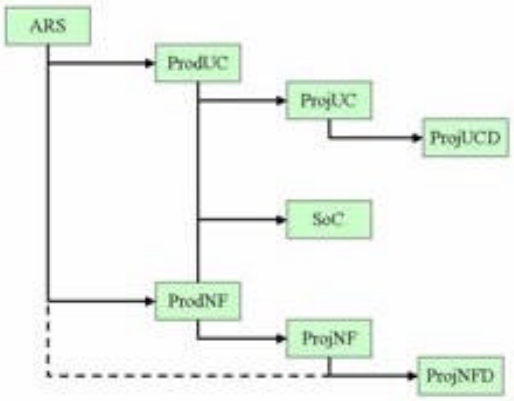


Figure 7.2 – Requirements structure

Basing the inspections on other artifacts than the test team gives more orthogonal approach. It's is a danger, if OORT are to be performed, based on the same artifacts as the test team uses. The PBR techniques tailored to UP uses checklist to go through the artifacts. At first look OORT seems superior to this technique when it comes to showing traceability between diagrams. PBR on the other hand seems more usable when applied to the whole model in an Object-Oriented process. The routines that are used at Ericsson

now touch some of the same principles when it comes to assigning different viewpoints (usability, testability, High Level). Though it doesn't use the principle of stakeholder (users, architect, tester and so on), it could easily be expanded with this. The stakeholders are also identified in the ARS, so it should be easy to identify them. Maybe the review and inspection team should look more on PBR tailored for UP for hints?

But again dropping the OORT for PBR would cause Ericsson to miss the whole concept of inspection based on traceability between diagrams. Experiment showed that OORT was valuable in a different way than the original techniques which are closer to PBR tailored to UP. Besides, it could be more effective to use PBR in the Requirements workflow since there are so many "perspectives" involved already. This is considered in the next chapter.

7.2.2 OORT in addition to current R&I process

An inspector can't inspect software by looking at it. It's invisible. He or she has to look at the representation of that object. In a more general sense, the logical entities making up a system, and the relationship between them, are collectively viewed as the architecture of the system.

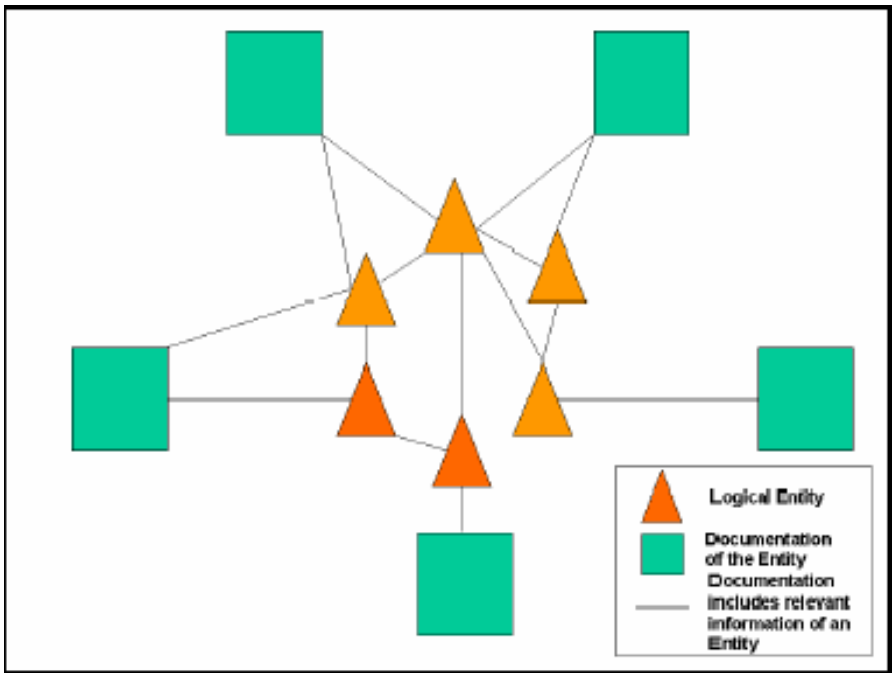


Figure 7.3 - Entities and the documentation of them

By looking at this, we see that it would mean inspecting only and one document, that one would miss seeing the wholeness of an entity. It's good to have reviews of single documents/diagrams to check for inconsistency with regards to the old document/diagram or for internal quality, but one should also inspect whole entities, discovering other defects and also finding defects of other character than the test team. This could be called an Architecture-driven inspection.

The Unified Process, for example, distinguishes between structural elements, such as subsystems or classes, and models that describe the structural elements, such as use-case or collaboration diagrams. Hence, the Unified Process uses the term "structural element" instead of logical entity and "model" instead of documentation of the logical entity. We decided to use the word "logical entity", since it best conveys the conceptual and invisible nature of software, and "documentation", since it best conveys the idea of something tangible that can be used for the purpose of inspection.

The architecture-centric solution for inspection organization is beneficial for three reasons: First, the set of information for each logical entity by definition is logically self-contained and conceptually complete. It therefore provides an inspector with all crucial information for performing a sound inspection and, at the same time, represents the appropriate set of information that is intellectually manageable. The latter prevents inspectors from being swamped with a lot of unnecessary information. Second, the architecture-centric approach is scalable. If the documentation of a logical entity is still too large, an inspection organizer can look at the substructure of the logical entity and choose an appropriate logical entity of smaller granularity. This process can be repeated until the right scope for a single inspection is determined. This method has been used with conventional methods, such as the Cleanroom Process [11]. But it’s not limited to conventional methods, it can also be tailored to Object-Oriented Development Processes.

Review guidelines in Sweden use architecture-centric scope when inspecting sub-systems, classes, processes and so on, they call it ”component review”. It is unclear in what degree this is adopted in Grimstad. Our suggestion is that this is also used with Use Case realizations. This comes from the nature of the OORT’s where structural elements are compared with dynamic artifacts as e.g. sequence diagrams.

Architecture centric approach in Ericsson’s context could be a little misleading, because architecture in Ericsson often is connected with the Analysis model and not the Design model that is our main target.

Architecture-centric software inspections in the context of the RUP can be organized around components, their interfaces, and their interactions. Components as seen from the development point of view are subsystems that have high internal cohesion and low external coupling and are reusable by other developers. A component as part of the architecture is best represented by multiple, coordinated architectural views. Just as the design model in the GPRS project is shown in different views. Inspections can then be made for what’s available at the moment from the different views connected to the subsystem or another logical entity.

So the Architecture-centric way of using OORT with the design model seems to make sense. What about the analysis model? It have to be investigated if not it is more effective to use PBR for inspections with the Analysis Model, because there are so many different views (perspectives) and the fact that state Diagram are never present in the Analysis Model.

If we were to use OORT with the Analysis model, then SAD and Design decisions are important documentation when it comes to using OORT on the Analysis Model.
OORT for Analysis Model can possible be used after these activities: ”Analysis of architecturally significant Use Case”(Sequence Diagrams are made) and ”Incorporate existing design elements”. They do perform Review architecture after these activities (at least according to the review guidelines from Sweden), where they ”reverse-engineer” against the Design Model and search for mismatches against requirements. The question is whether these inspections are based on what to defects to find and not how to find them. Remember that one of the positive comments from the experiment was that OORT helped them find defects that they couldn’t see at first sight.

For use of OORT in the Deployment and Physical View of the Analysis Model, the deployment View in the SAD has documentation on architecturally significant process environment blocks and applications that will be further modeled by designers.
For use of OORT in the Process View of the Analysis Model, the Process View in the SAD has documentation on how Model elements are distributed among processes and process lifecycles as well as definitions of concurrency requirements. The process Model in Rose also connects to this.

During the Design centric activities, State Diagrams are possibly made in ”Identify and create design elements” and Sequence Diagrams are made in ”Use Case design”.

Understanding that the Architecture-centric approach would make it possible to use OORT on the Design Model and maybe also on the Analysis Model, how could we find the entities in the system we should focus on? Can VOPC in the analysis model show architectural important entities? The UCS and the flows therein are used as input for that artifact.

There seems to be two different ways of finding the entities of a system. One attempt is to locate them in the VOPC of the Analysis Model, the other is to use ”only” the subsystems and the blocks therein as entities. Sometimes the VOPC are not very updated since doing so sometimes is postponed. If so, one possibly has to look in the SAD.

If VOPC is used one can also look at the sequence diagrams in help for finding out how entities relates to each other. This can then be used as input for tailoring the OORT’s.

It remains unclear when inspections should occur. In other words, what conditions or events trigger the inspection of the diagrams and related artifacts for a given logical entity? Since RUP is an iterative and incremental process, driven by use-cases, the various models defined by RUP are not developed in sequence (as in the classic waterfall model) but are rather elaborated incrementally over time. In other words, each iteration adds a slice of information to the various models as part of the elaboration of the driving use-case. Since we are advocating an architecture-centric organization of inspections, this means that an inspection of a logical entity (such as a subsystem or a class) should be triggered when all the relevant information is ready for examination (including analysis, design and implementation information).

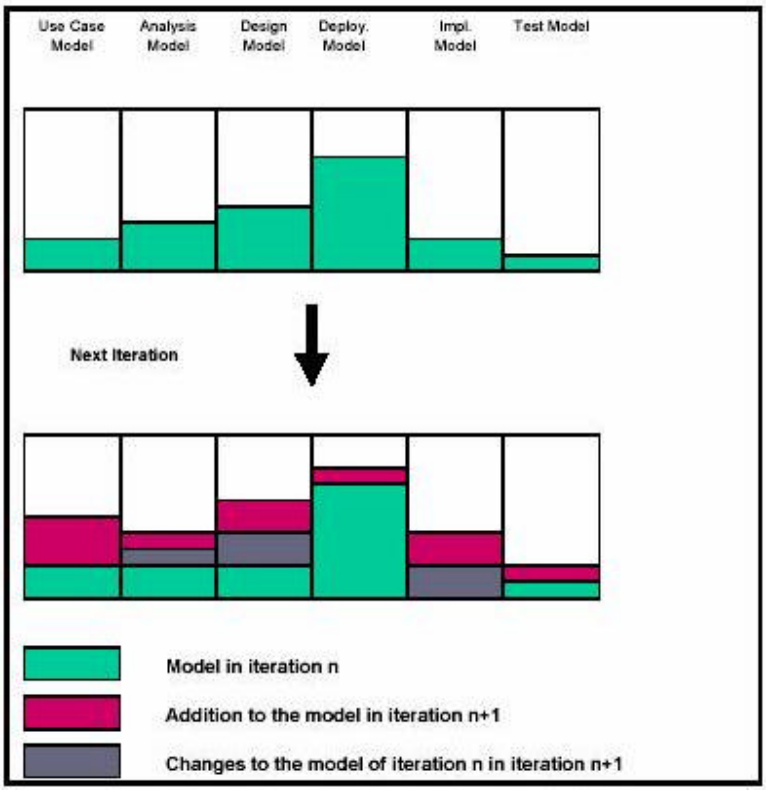


Figure 7.4 - Incremental development

At first sight this might appear to imply that all inspections will defer to the end of the entire project, but this is not the case in RUP for two reasons. First, because RUP follows an incremental approach, analysis, design and implementation determines the availability of the necessary inception information is not the level of abstraction (e.g.. analysis or design level) as in traditional methods, but the completion of all the functional slices (e.g.. the use-cases) affecting the logical entity concerned. Assuming that most logical entities are only involved in a subset of the use-cases, it follows that they will become available for inspection as soon as their functionality has been covered. The order in which use-case are developed is therefore the primary factor when finding out if the documentation of a logical entity is sufficient for inspection. For large systems one could perform intermediate inspections, based on functionality covered up to that point. When the Use Cases are developed, is planned in Iteration and Integration Plans.

The logical entities themselves as well as the priority of use-cases can be identified at a early stage of RUP development project, since the architecture is stabilized in the early development phases.

7.3 Organizing the OORT inspections

Since RUP develops incrementally it means that both analysis, design and implementation determine the availability of the necessary information for inspection not based on the level of abstraction but on the completion of all functional slices (e.g.. the UC's) affecting the logical entity. The logical entities can be described as figure 7.1 shows.

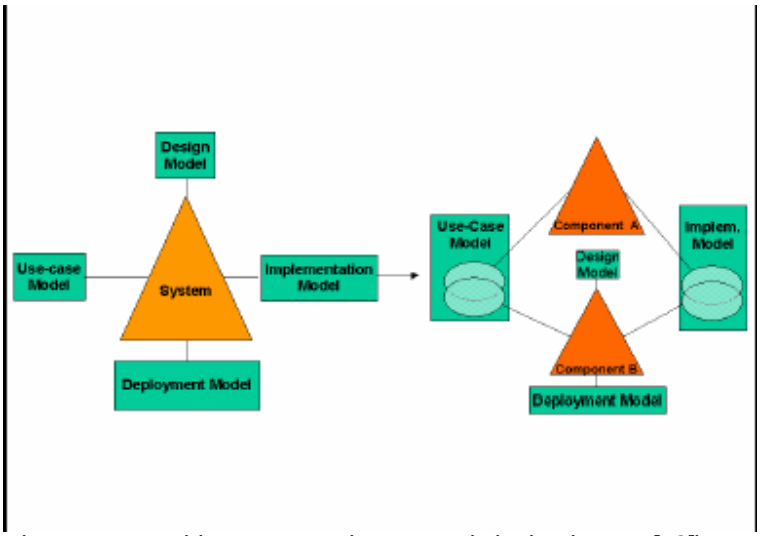


Figure 7.5 - Architecture-centric approach (Laitenberger [18])

This is very meaningful in an Ericsson context, since architecture already is a considerable input for designers (referring to SAD). As mentioned, the inspections would not be based on the increments, but on when the logical entity is complete enough to be considered as a target for inspection. Complete enough would mean that the connected UC's are implemented. This allows inspections to be performed early, since the architecture is set on a very early stage of the process. Very large logical entities could then be target for intermediate inspections based on the parts of the entity that have been implemented to that point.

The selection of artifacts should be done by sub-system responsible, since he/she should have an overview of the system. Then it should be delivered to Use Case team leader for review. The sub-system responsible should plan the inspections together with the Use Case team leader, consulting Architect whenever needed. The Iteration plan and the Integration plan determine when and which UC's are implemented. The same UC's that make the fundament for when an inspection should be conducted on the logical entity also lists references to the requirements for the logical entity. The FIS documents also list what artifacts that are affected by implementation from the UC. So the documents that help finding the set of information to an inspection are: UCS, FIS and SAD. They are sometimes leading to other documents such as SoC/Standards and Supplementary Specifications. The documents that determine when an inspection should be performed are ARS (it lists the UC´s), iteration plan and integration plan. Inspections should finally be documented in the R&I plan by the team leader, assisted by sub-system responsible.

Control of how often OORT inspections are to be performed are then given to the sub-system responsible.

Finally it has to be identified similar artifacts as those used in the original OORT's. This should be done before OORT are taken into use in the development process, but to some extent it may be needed to do further tailoring, though it should be minimal. So when the logical entity is identified and the artifacts are on place on the planned time for inspections, it should be easy to pick out a subset of the OORT's to be used on the logical entity. I.e. if a logical entity at a given point that inspection is planned, has not implemented state-diagrams yet, the OORT's regarding State diagrams are not used, but could be used in a later inspection. The team leader should be aware of and control that the artifacts that are planned used for the inspection are in sync in time for the inspection, labeling them in ClearCase. He or she should also be responsible for giving feedback if review and inspections plans are to be changed. This should be consulted with the sub-system responsible.

Because of this the inspections are not dependent on unstable increment deadlines and avoid the time-pressure problem. Inspections can be performed whenever after an entity's functionality up to a given level has been implemented.

7.4 OORT inspection guidelines

Regarding the techniques we assume that the best would be to use OORT as close to the original set as possible. It should maybe be postponed until more industrial experiments are conducted. Anyway the OORT has proven their strength in all the experiments it has been through so far. One could also hope for a use of the architecture centric approach for organizing inspections with OORT's from its original authors (Laitenberger) [12].

So what document/diagrams could replace the ones in the original OORT's? Remember this is only meant as a proposal, or means to discussion. The OORT's should be throughout examined by someone with great knowledge of the model and the system at the GSN project at Ericsson.

When adjusting the OORT's to be used at Ericsson, one has to look at how they originally were intended to be used.
The original OORT's rely on two artifacts from requirements, and that is Requirement Description and Use Case diagrams.

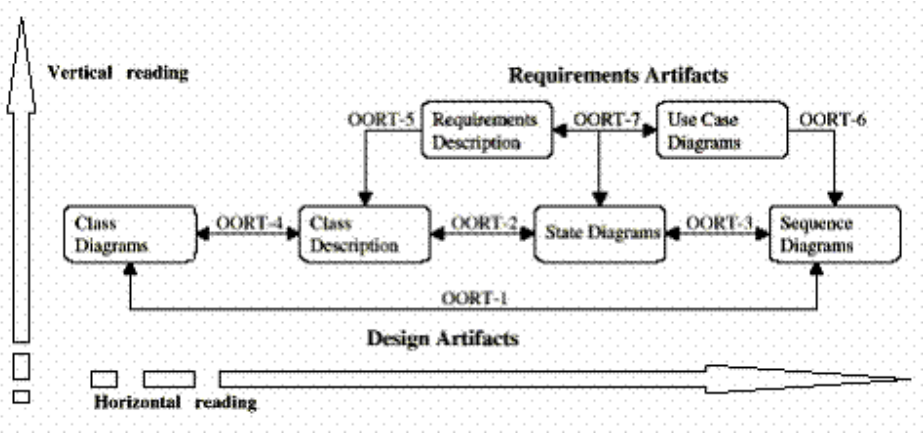


Figure 7.6 – Vertical Reading and Horizontal Reading

The Requirements Description is supposed to offer certain qualities. It should be possible to mark candidate classes/objects/attributes, services and constraints/conditions in a document. It should be possible to read the functional requirements to determine the possible states of the object, which states are adjacent to each other, and which events/actions cause the state changes.

The Use Cases (the other requirement artifact) should be used to determine *what* causes the state changes. In addition it should be possible to mark system concepts and services and the data necessary to achieve such services.

If these requirements qualities are present, the requirement part of the OORT's should be in place.

The high level design documents/diagrams that could be used as input for those OORT's that uses them are:
Use Case Specification, Supplementary Specification. These two could be filtered through (several) FIS documents to guide the organizer in hiding requirements information in these rather huge documents that does not apply to the inspection. If further requirements are needed according to the OORT's, one could look in the references f the UCS and the SS's to find requirements on a even higher level. This could be reference to ARS, SoC (Standards), CR's, other UCS's and SS's.

Low – level design are easier to identify with the OORT's.
The low level documents/diagrams that could be used as input for those OORT's that uses them are:

Class diagrams and state-diagrams in the sub-systems, Class diagrams and state diagrams in the Blocks. Possible Classes/Interfaces in the units. Use Case Realizations as Sequence diagrams divided into Main Flows, Alternative Flows and Exceptional Flows. Class descriptions in the packages, in the Logical View of the Design Model.

The Sub-systems and the Blocks even have one class diagram that are functional and one that is non-functional that could be used against the UCS and the SS'. In addition there is an overview diagram. For further details, see Appendix B.IV.

The OORT's would then look something like this, hiding the questions that could need a closer review. Look in the original set of OORT's (Appendix C) for the questions:

- OORT-1 Sequence Diagram(UCR) x Class Diagram(Overview-Subsystem)
- OORT-2 State Diagram(Sub-system) x Class description (Package)
- OORT-3 Sequence Diagram(UCR) x State Diagram (Sub-system)
- OORT-4 Class Diagram(Overview-Subsystem) x Class Description (Package)
- OORT-5 Class Description(Package) x UCS & SS other req.(Through FIS('s))
- OORT-6 Sequence Diagram(UCR) x UCS
- OORT-7 State Diagram(Sub-system) x (UCS & SS other req.(Through FIS('s))

We have not had the time to look closer at the questions, and most likely there are information missing in these artifacts that are requested in the techniques. The techniques as we have listed them here are the ideal set, assuming that all the wanted information is available. Also the original purpose of horizontal and vertical reading is kept here.
It has to be developed a way of answering the questions regarding the Classes. Since this cannot be done on print-outs it should be recorded in some other way. Maybe bit could be made logs that could easily be filled out.

The OORT's should be extended with guidelines that cover other models connected to the entity. This is somewhat covered by the existing R& I method, but then use-case driven and not focused on the architectural-centric approach, maybe except reviews of components, possibly missing important information. But of course, it is partly covered, since e.g.. the deployment-represent attend to multiple reviews, in sum covering some of the entity. It could also be considered to use other reading techniques on these models, and also there already are other methods in use in Grimstad for doing this. This has been partly discussed already in chapter 6. To sum it up, support material should include:

- Guidelines on when and based on what in the process OORT's could be used
- Guidelines on roles/workers
- Guidelines on relevant artifacts
- Guidelines on suited OORT's

Suggestions to the current R&I process would be:

- Use more detailed/different defect logs
- Use web based logging

7.5 Summary

OORT coupled with iterations and deadlines is not preferable because of little overview on the needed artifacts, and trouble with time-pressure. Basing OORT inspections on logical entities in the architecture on the other hand, makes it easy to provide the needed artifacts, even if they are spread outside the scope of a Use Case team.
At Ericsson, Grimstad, possible entities could be sub-systems and blocks or they could be identified using the VOPC in the Analysis Model, since it is meant to show architecturally significant classes.

OORT could also be used with the Analysis Model as a target, but it should be considered if not PBR should be used on that model instead, because of its many “perspectives” and the fact that state-diagrams are never present there.

The time for performing OORT inspections based on entities are not limited to the abstraction of the models. Because of this, it can be performed very early in the phases. The architecture, being laid on an early stage makes the architecure-centric approach very suitable to be used with early fault-detection methods in an Object-oriented development process.

8 Conclusion

In our Thesis we have described an improved inspection process for Ericsson, Grimstad. We have thoroughly studied the system, the development process and the roles at Ericsson so that our suggestion would fit their needs. Part of our Thesis was to conduct an experiment, comparing the two techniques. Results from it shows that the OORT's focuses the inspectors in a development process on the Model and help finding defects of different character than their current R&I method. The techniques also lead the inspector to find more subtle defects. Ericsson's current R&I method find more defects of technical value. This makes the two techniques complementary.

The reading techniques had to be suited to the Ericsson context. The target workflow is the Analysis & Design. It should mainly be used in the Design Model but also the Analysis Model could be target for an OORT inspection. We wanted to base the OORT inspections on other premises than the current R&I method, making the two techniques supplementary also on a process level. We found out that the OORT inspections would fit into RUP, if the architecture-centric approach is used. This approach sees the system as an entity, possible divided into several sub-entities. Each entity is self-contained with a set of information that is conceptually whole and logically complete. The inspection organizer will be supplied with intellectually manageable and crucial information for performing an inspection. Further on the approach is scalable, which means that if the entity is too large, an inspection organizer can look at the substructure of the logical entity and choose an appropriate entity of smaller granularity. In addition, OORT-inspections are not restricted to a deadline, since it is not performed because of implementation of a new functionality. Thus avoiding time-pressure and poor preparations, as a result of prioritising design work. When a system entity's functionality have been fully or partly covered up, a OORT inspection can be performed whenever after this level has been reached, assuming that the actual artifacts are labelled in e.g.. a personal view in ClearCase. Example of an entity is a subsystem.

As far as we can see, Ericsson would profit from implementing OORT in their inspection process. Papers and articles have been written on how UML designs often are filled with more defects, though of smaller size, than conventional design. This could speak for that. Ericsson could have an increased cost when implementing the new routine in their organization, but in the long run they would most likely gain from it. The OORT's needs some adjustments to the Ericsson context, but should be kept as close to the original set as possible, maintaining its original strengths. More detailed guidelines should also be written down. It should be investigated if the Architectural views in RUP also could be used when detecting entity's and it should be experimented with use of OORT and also PBR in the requirement workflow. This Thesis could also be a valuable input for the Method & Tool team at Ericsson, Grimstad.

Further on, guidelines for the new technique must be developed and more industrial experiments should be performed with the OORT's. Possibly one should also compare PBR and OORT. The OORT's themselves should also be generalized to fit modern Object-Oriented Development Processes as RUP, Extreme Programming and others.

9 Abbreviations

ARS	Application Requirement Specification
CBR	Check-list Based Reading
FIS	Feature Impact Study
GPRS	General Packet Radio Service
GSN	GPRS Support Node
OORT	Object Oriented Reading Techniques
UC	Use Case
UCD	Use Case Diagram
UCS	Use Case Specification
UCR	Use Case Realization
UML	Unified Modeling Language
PBR	Perspective Based Reading
ProdUC	Product Use Case
ProdNF	Product Non-functional Requirement
ProjUC	Project Use Case
ProjUCD	Project Use Case Detail
ProjNF	Project Non-functional Requirement
R&I	Review and Inspection
RUP	Rational Unified Process
SAD	System Architectural Description
SBR	Scenario Based Reading
SGSN	Serving GPRS Support Node
SoC	State of Compliance
SS	Supplementary Specification
UP	Unified Process

10 References

[1] Profit - Process Improvement for IT industry: “PROFIT – Generell presentasjon”, 2001, Foil set, 13 slides.

[2] Martin Fowler w/Kendall Scott: "UML Distilled, Second Edition", Addison Wesley, New Jersey, USA, 2000 ISBN 0-201-65783-X.

[3] Philippe Krauten: “The Rational Unified Process, an introduction”, Addison Wesley, New Jersey, USA, 1999 ISBN 0-201-60459-0.

[4] Guilherme H. Travassos, Forest Shull, Jeff Carver, Victor, R. Basili: “Reading Techniques for OO Design Inspections”, SEW99_paper, 10 pages.

[5] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén : “Experimentation In Software Engineering – An Introduction”, Kluwer Academic Publishers, Boston, USA, 2000, 205 p. ISBN 0-7923-8682-5.

[6] Tayyaba Arif, Lars Christian Hegde, NTNU: “Inspection of Object Oriented Construction”, Pre Diploma Thesis, 2001, 135 pages.

[7] Guilherme H. Travassos, Forest Shull, Michael Fredericks, Victor, R. Basili: “Detecting Defects in Object Oriented Designs: Using reading techniques to increase software quality”, 1999, paper 10 pages.

[8] Walcélio Melo, Forest Shull, Guilherme H. Travassos: “Software Review Guidelines”, 2001

[9] Reidar Conradi, NTNU: "Preliminary NTNU report of the OO Reading Techniques (OORT) exercise in course 7038 on Programming Quality and Process Improvement, spring 2000, v1.12", 2001

[10] Kaiserslauten: “Defect Cost Reduction”, 2001, 21 pages.

[11] O. Laitenberger, Cost-effective Detection of Software Defects through Perspective-based Inspection. PhD Thesis, University of Kaiserslautern, 2000.

[12] Oliver Laitenberger, Kristin Kohler, Colin Atkinson: **“**Architechture-centric Inspection for the Unified Development Process (UP)”

[13] Tom Gilb, Dorothy Graham: "Software inspection", Addison Wesley, London, UK, 1993. ISBN 0-201-63181-4

[14] Magne Ribe: “Review & Inspection Training”, 2001, foils, 85 p.

[15] Reidar Conradi, Amarjit Singh Marjara, Øivind Hantho, Torbjørn Frotveit, and Børge Skåtevik: "A study of inspections and testing at Ericsson, Norway", 2001, paper 20 pages

[16] Guilherme H. Travassos, Forest Shull, Jeff Carver, Victor, R. Basili, UMD, and Reidar Conradi, NTNU, p.t. Univ. Maryland: "Fag 45038 Programvarekvalitet og prosessforbedring, IDI, NTNU, Trondheim, våren 2000, 21.feb.2000-Object-Oriented-Reading Techniques (OORTs) for design Documents: general and technical aspects (v1.4)", foils, 64 p.

[17] Olivier Laitenberger, Colin Atkinson: "Generalized Perspective-based Inspection to handle Object-Oriented Development Artifacts", Proceedings of ICSE 1999, pages 494 – 503.

[18] Oliver Laitenberger, Colin Atkison, and Khaled El Emam: "Using Inspection Technology in Object-oriented Development Projects", June 2000

[19] M.E. Fagan, "Design and code inspections to reduce errors in program development", IBM Systems Journal 15 (No. 3, 1976), pp. 182-211

[20] F. Shull, J. Carver, G. Travassos, "An Empirical Methodology for Introducing Software Processes", (ESEC01), 9 pages.

[21] O. Laitenberger, K. Emam, T. Harbich, "An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-based Reading of Code Documents", (isern-99-01), 57 pages.

[22] The intranet at Ericsson. (Not public)

[23] http://www.esernet.org

[24] http://www.iese.fhg.de/ISERN/

[25] Travassos, Shull & Carver, "Working with UML: A Software Design Process based on Inspections for the Unified Modeling Language" - Utdrag fra bok

[26] Travassos, Shull & Carver, "An Empirical Methodology for Introducing SoftwareProcesses"

[27] Travassos, Shull & Carver, "Evolving A Process for Inspecting OO Designs"

[28] Travassos, Shull & Carver, "A Family of Reading Techniques for OO Design Inspections"

[29] Software Engineering Group, NTNU

[30] Phillipe Kruchten, "A Rational Development Process"

Appendix A - Thesis definition

Supervisors: Parastoo Mohagheghi (ETO S/R/Z) in contact with Prof. Reidar Conradi at NTNU. Gunhild Sørensen Lundvall and Magne Ribe from ETO will also support the student(s).

Student(s): Geir Arne Bunde and Anders Pedersen

Responsible line manager: Fritz Ekløff.

Thesis Title: Defect reduction by Improving Inspection of UML diagrams in the GPRS project.

Subtitle: A study of available techniques and state-of-the-practice in the GPRS project for inspection of UML diagrams. Experimenting a suggested method to reduce defect cost by early detection of defects.

Background: Some defects can not be found by testing. Further defects found late are expensive to correct. Thus techniques for early defect detection are needed. Inspections performed early in the development are one of those techniques. ESERNET (Experimental Software Engineering Research Network) is a network started on 01.08.2001 as one of the EU R&D programs where NTNU and UiO are also participating. One of the project goals is to perform experiments in the industry and among students, especially in the fields of component-based development, inspections and testing. Experiments are either industrial survey (description of the state-of-the-practice), benchmarking a standard inspection technique or controlled experiments where a baseline also exists. Organisations can choose the experiment option based on the effort and anticipated benefits; as a controlled experiment needs more effort than benchmarking which in turn needs more effort than a state-of-the-practice study. The results will be used for product and process improvement in the organisation and be part of the ESERNET experiments to learn how to conduct studies to be able to react to challenges in the future.

The GPRS project at Ericsson takes advantage of the Rational Unified Process and UML diagrams for requirement engineering, analysis, design and test. While the GSN RUP adaptation describes the artifacts that should be produced in different stages of the project life cycle, inspection of artifacts has received less attention. Software reading techniques try to increase the effectiveness of inspections by providing guidelines that can be used by reviewers of software artifacts. Object-oriented Reading Techniques (OORT) is a method for inspecting UML diagrams and their relationships and consistency.

Early fault detection is the main focus of a recently started project in Grimstad that focuses on inspection improvement. This thesis's results can be used in the project and integrated into the existing inspection processes.

Thesis definition

- Thesis goals are:
- Study the inspection techniques for UML diagrams.
- Study the state-of-the-practice in the GPRS project for inspection of UML diagrams.

Design and conduct an experiment where the subject is to compare the existing inspection technique in the GPRS project for UML diagrams with an assumed improved variant. The experiment is based on results from a pre-diploma thesis written at NTNU and an experiment done during spring 2002 where the participants are students from a course at NTNU. The students from HiA and NTNU will cooperate for design of these experiments. The goal is to learn how to conduct and analyse industrial experiments and to evaluate the suggested improvement. Participants may be employees of Ericsson or students from HiA.

- Based on the studies and the experiment results, suggest improvements to the inspection techniques for UML diagrams in the GPRS project. Develop guidelines that may be used by reviewers during inspections.

Competence:

- Object-Oriented requirement specification, analysis and design using UML.
- Basic knowledge of the Rational Unified Process.

During the work, the students will learn more about OO design, inspection techniques and goals, the Rational Unified Process, performing experimental studies in the industry and analysing the results.

Security: The students should have access to the GPRS model and process documentation to study the current inspection techniques.

Originality, IPR and reuse:

Limitations:

Activities: As described in the thesis definition.

Prerequisites:

Working place and conditions: The students need access to the file system at Ericsson during the project time (UML model and documents). Experiments may be done at Ericsson or HiA.

Budget and funding: The SIMULA research lab at IT-fornebu can finance experiment costs. A possible budget is 10 persons a 10 hours (a 500 NOK?) = 50000 NOK, including training in relevant techniques.

References:

- http://www.esernet.org/

- http://www.idi.ntnu.no/grupper/su/

- Material on ESERNET project and OORT are available by asking the author.

Appendix B - Ericsson baselining

Confidential

Because of the material included in this Appendix cannot be published; it was removed from the thesis. The results from the baselining and the other material can be made available on request and approval from Ericsson.

Appendix C – Initial OORT

The initial set of reading techniques, improved by the NTNU students, before any changes were implemented.

OORT-1: Sequence Diagram x Class Diagram

Inputs:
1. A class diagram, possibly in several packages.
2. Sequence diagrams.

Outputs:
1. Annotated versions of above diagrams.
2. Discrepancy reports.

Goal: To verify that the class diagram for the system describes classes and their relationships in such a way that the behaviors specified in the sequence diagrams are correctly captured

Instructions:
Do steps R1.1 and R1.2.

Step R1.1: From a sequence diagram – identify system objects, system services, and conditions.

Inputs:
1. Sequence diagram (SqD).

Outputs:
1. System objects, classes and actors (underlined with **blue** on SqD);
2. System services (underlined with **green** on SqD);
3. Constraints/conditions on the messages/services (circled in **yellow** on SqD).
I.e., a marked-up SqD is produced, and will be used in R1.2.

Instructions: – matches outputs above.

Q11.a: Underline system objects, classes and actors in **blue** on SqD.

Q11.b: Underline system services in **green** on SqD.

Q11.c: Circle constraints/conditions on messages/services in **yellow** on SqD.

Step R1.2: Check related class diagrams, to see if all system objects are covered.

Inputs:
1. Marked up sequence diagrams (SqDs) – from R1.1.
2. Class diagrams (CDs).

Outputs:
1. Discrepancy reports.

Instructions (as questions – here and after**):**

Q12.a: Can every object/class/actor in the SqD be found in the CDia?
Possible *[inconsistency*?]

Q12.b: Can every service/message in the SqD be found in the CDia, and with proper parameters? *[inconsistency*?]

Q12.c: Are all system services covered by (low-level) messages in the SqD? Possible *[omission*?]

Q12.d: Is there an association or other relationship between two classes in case of message exchanges? *[omission*?]

Q12.e: Is there a mismatch in behavior arguments or in how constraints / conditions are formulated between the two documents? *[inconsistency?]*

Q12.f: Can the **constraints** from the SqD in R1.1 be fulfilled?
- E.g. Number of objects that can receive a message (check cardinality in CDia)?
- E.g. Range of data values?
- E.g. Dependencies between data or objects?
- E.g. Timing constraints?

Report any problems. *[inconsistency*?]

Q12.g: **Overall design comment**s, based on own experience, domain knowledge, and understanding:
- E.g. Do the messages and their parameters make sense for this object?
- E.g. Are the stated conditions appropriate?
- E.g. Are all necessary attributes defined?
- E.g. Do the defined attributes/functions on a class make sense?
- E.g. Do the classes/attributes/functions have meaningful names?
- E.g. Are class relationships reasonable and of correct type? (ex. association vs. composition relationships).

Report any problems. *[incorrect fact*?]

OORT-2: State Diagram x Class Description

Inputs:
1. A set of class descriptions.
2. A set of state diagrams for the system objects.

Outputs:
1. Discrepancy reports.

Goal: To verify that the classes are defined, so that they can capture the functionality specified by the state diagram.

Instructions:
Repeat steps R2.1 – R2.3 for each state diagram (StD).

Step R2.1: Read state diagram to understand possible states and their transition.

Inputs:
1. State diagram (StD).
2. Set of class descriptions (CDe).

Outputs:
1. Object states (marked in **blue** on StD).
2. Transition actions/conditions (marked in **green** on StD).
I.e., a marked-up state diagram is produced, used in R2.2 and R2.3.
3. Discrepancy reports.

Instructions:

Q21.a: Identify the actual class from the state diagram. Missing? *[omission*?]

Q21.b: Underline the name of each object state (by **blue** pen).

Q21.c: Underline the transition actions/conditions (by **green** pen).

Q21.d: Can you understand the object's behavior from Q21.b-c above? *[ambiguity*?]

Step R2.2: Identify the associated class, and its attributes and behavior.

Inputs: partly from state diagram (StD)
1. Set of class descriptions (CDe).
2. Object states (marked in **blue** on StD – from R2.1).
3. Transition actions/conditions (marked in **green** on StD – from R2.1).

Outputs:
1. Discrepancy reports.

Instructions:

Q22.a: In the CDe, identify the class being modeled by this state diagram.
Missing? *[omission*?]

Q22.b: Find out how a **blue** state is represented, i.e. has the class captured each modeled state in a unique way?
E.g. by an explicit attribute.
E.g. by an implicit attribute (merely via control flow).
E.g. by a combination of attributes.
E.g. by subtyping of the actual object (consult the class hierarchy).
Report the result. *[inconsistency?* or *ambiguity*?]

Q22.c: Are all **green** transition actions/conditions covered by class behavior?
If not: error. *[inconsistency*?]

Q22.d: Are **green** transition conditions using object data that are defined as class attributes with matching names?
If not: error. *[inconsistency*?]

Step R2.3: Compare class diagram to state diagram.

Inputs: from state diagram (StD)
2. Object states (marked in **blue** on StD – from R2.1).
3. Transition actions and conditions (marked in **green** on StD – from R2.1).

Outputs:
1. Discrepancy reports.

Instructions:

Q23.a: From your domain knowledge, are all relevant states defined in the StD?
[incorrect fact?]

Q23.b: For each unmarked state, assess if it is appropriate and essential:
[incorrect fact? or *extraneous*?]

Q23.c: For each unmarked transition action/condition: here is missing information.
[inconsistency?]

OORT-3 Sequence Diagram x State Diagram

Inputs:
1. A set of sequence diagrams.
2. A set of state diagrams for several objects.

Outputs:
1. Discrepancy reports.

Goal: To verify that every state transition for an object can be achieved by the messages sent and received by that object.

Instructions:
Repeat steps R3.1 – R3.3 for each state diagram (StD).

Step R3.1: Read the state diagram to understand the possible object states, their transitions and corresponding actions.

Inputs:
1. Given state diagram (StD).

Outputs:
1. Marked-up state diagram (StD), with transition actions labeled in **gree**n.
2. Discrepancy reports.

Instructions:

Q31.a: Determine which class is being modeled. Missing? *[omission*?]

Q31.b: Trace all transitions from the start state to the end state, and mark corresponding actions with a unique name (A1, A2 etc.) with a **green** pen.

Q31.c: In general, do these transitions/actions and states make sense and are they understandable for such an object? *[ambiguity*?]

Step R3.2: Read the sequence diagrams to understand how the transition actions are achieved by messages sent to/from the relevant object.

Inputs:
1. Marked-up state diagram (StD) (w/ transition actions in **green –** from R3.1).
2. Set of sequence diagrams (SqD).

Outputs:
1. Marked-up sequence diagrams (SqD), with matching object messages labeled in **gree**n.
2. Discrepancy reports.

Instructions:

Q32.a: Pick the relevant subset of SqDs concerning this state diagram (StD)
Is there a problem to identify these? *[omission?* or *extraneous*?]

For each relevant sequence diagram (SqD) do below points Q32.b-e:

Q32.b: Read the sequence diagram to identify the associated system service and its messages.

Q32.c: Identify the object states in the StD, being semantically related to the actual system service.

Q32.d: Map message arrows (one or many) in the SqD to state transitions in the StD. Are there "enough" messages to accomplish a given transition? *[omission*?]
Mark related SqD-messages and StD-transitions with a **green** star.

Q32.e: Look for constraints and conditions on the above SqD-messages. Check if the same constraint/condition information stands in both diagrams. *[inconsistency*?]
Such SqD-information may be correspondingly expressed in the StD by:
1) State information (e.g. t>0),
2) Transition information (what occurs when t>0?),
3) Nothing (not relevant for StD).

Step R3.3: Review the marked-up diagrams to make sure that all transition actions are accounted for.

Inputs:
1. Transaction actions on the given StD (labeled in **green –** from R3.1)
2. Object messages on the SqD (labeled in **green –** from R3.2)

Outputs:
1. Discrepancy reports.

Instructions:

Q33.a: Look for unlabeled **transaction actions** in the StD, i.e. those not implemented by available messages in the SqD (cf. Q32.d). Report these. *[inconsistency*?]

Q33.b: Are the **event order** the same in the StD and SqD, i.e. check if labeled messages/transitions in the SqD appear in logical order? *[inconsistency*?]
E.g. that action Ax on a later transition in the StD actually occurs after an action Ay on an earlier transition.

OORT-4: Class Diagram x Class Description

Inputs:
1. A class diagram (CDia), possibly in several packages.
2. A set of class descriptions (CDe).

Outputs:
1. Discrepancy reports.

Goal: To verify that the detailed descriptions of classes contain all the information necessary according to the class diagram, and that the description of classes make semantic sense.

Instructions:
Repeat steps R4.1 and R4.2 for each class in the class diagram (CDia).

Step R4.1: Read the class diagram to understand the necessary properties.

Inputs:
1. Given class from class diagram (CDia).
2. A set of class descriptions (CDe).

Outputs:
1. Discrepancy reports.

Instructions:

Q41.a: Is there a CDe for this class? *[omission*?] Mark with a star ("*") in **blue** on the CDe when found, see R4.2.

Q41.b: Is the name and textual description of this class meaningful in the CDe? *[ambiguity*?]

Q41.c: Are **attributes** and their types consistent between the CDia and CDe? *[inconsistency*?]

Q41.d: Can this class meaningfully contain all these attributes and with given types? *[ambiguity?* or *incorrect fact*?]

Q41.e: On **behavior and constraint**s:

- E.g. Check consistency for behavior and constraints between the CDia and CDe.
- E.g. Are behaviors in the CDe described at the same level of detail / pseudocode? *[inconsistency*?]
- E.g. In general, should this class really contain all these behaviors and constraints? *[incorrect fact*?]

E.g. Do the behaviors and constraints in the CDe use available behaviors or attributes from elsewhere, and are they defined? *[omission?* or *ambiguity*?]

E.g. Do the behaviors and constraints in the CDe rely "excessively" on attributes in remote classes? I.e. too high coupling? *[miscellaneous*?]

Q41.f: In case of use of **inheritance** in the CD:

E.g. Is inheritance also included in the CDe? *[omission*?]

E.g. In general, is it "meaningful" for the given class be a supertype/subtype of the given subclasses/superclass? *[miscellaneous*?]

Q41.g: Check that all **relationships** are correctly described:

E.g. Do they have the right cardinalities, and are they also defined in the CDe? *[inconsistency*?]

E.g. We object roles in the CDia also defined in the CDe? *[inconsistency*?]

E.g. I s the correct graphical notation used in the CDia? *[inconsistency*?]

E.g. In general, do the stated relationships "make sense", such as composition vs. aggregation vs. association vs. inheritance etc.? *[miscellaneous*?]

E.g. Is an attribute used to represent a relationship, and does this have the right type (a reference or sets of references)? *[inconsistency*?]

Step R4.2: Review the class for extraneous information.

Inputs:
1. A set of class descriptions (CDe).

Outputs:
1. Discrepancy reports.

Instructions:

Q42.a: Are there any unstarred (i.e. superfluous) classes in the CDe? *[extraneous*?]

OORT-5: Class Description x Requirement Description

Inputs:
1. A set of requirement descriptions (RD), mainly functional.
2. A set of class descriptions (CDe).

Outputs:
1. Discrepancy reports.

Goal: To verify that the concepts and services that are described by the functional requirements are captured by the class descriptions.

Instructions:
Do steps R5.1 - R5.3.

Step R5.1: Read the requirements to understand the functionality described.

Inputs:
1. Set of requirement descriptions (RD).
2. Set of class descriptions (CDe).

Outputs:
1. Candidate classes/objects/attributes (marked in **blue** in RD).
2. Candidate services (marked in **green** in RD).
3. Constraints or conditions on services (marked in **yellow** in RD).
I.e., a marked-up RD is produced, used in R5.2and R5.3below.

Instructions:
Q51.a: Find the nouns, being candidates for classes/objects/attributes. Underline with a **blue** pen.

Q51.b: Find the verbs or action descriptions, being candidates for services or behaviors. Underline with a **green** pen.

Q51.c: Look for constraints and conditions on nouns/verbs above, e.g. for relationships, limiting quantities, or non-functional requirements. Underline with a **yellow** pen.

Step R5.2: Compare the class description to the requirements

Inputs:
1. Set of marked-up requirement descriptions (RD) **–** from R5.1
2. Set of class descriptions (CDe).

Outputs:
1. Corresponding concepts have been marked on the RD and CDe.
2. Discrepancy reports.

Instructions:

Q52.a: For each **gree**n-underlined verb/action in the RD:

E.g. Find associated behavior(s) in the CDe.

E.g. Do the classes/objects receive the right information to accomplish their required behavior, and are appropriate results produced? *[incorrect fact*?]

Q52.b: For each **blu**e-underlined noun/concept in the RD, try to find an associated class in the CDe, and mark both with a **blue** star ("*").

E.g. Does the class description contain sufficient and clear information for this concept, and does the class name resemble the noun you had marked? *[ambiguous*?]

E.g. Does the class encapsulate related **(blu**e-marked) attributes, and does the class encapsulate related **(gree**n-marked) behavior, and are all identified constraints and conditions for this class described in the RD? *[omission*?]

Q52.c: For each remaining, **blu**e-underlined noun/concept in the RD, try to find a matching attribute in the CDe, and mark both with a **blue** star ("*").

E.g. In general, is the CDe using appropriate types to represent information from the RD, and are the **(yello**w-underlined) constraints and conditions on these attributes also contained in the CDe? *[incorrect fact*?]

Step R5.3: Review the Class Descriptions and Requirement Documents to ensure that all concepts mutually correspond.

Inputs:
1. Set of marked-up requirement descriptions (RD) – from R5.1.
2. Set of marked-up class descriptions (CDe) – from R5.2.

Outputs:
1. Discrepancy reports.

Instructions:

Q53.a: Are there still unstarred **blu**e-underlined nouns or **gree**n-underlined activities in the RD, i.e. not being represented in the CDe? *[omission*?]
Note: some RD-concepts may have been used just for explanation.

OORT-6: Sequence Diagram x Use Case Diagram

Inputs:
1. A use case diagram (UC) for a part of the system, with its services.
2. One or more sequence diagrams (SqD) for relevant system objects and services.
3. A set of associated class descriptions (CDe).

Outputs:
1. Discrepancy reports.

Goal: To verify that sequence diagrams describe an appropriate combination of objects and messages that capture the functionality from the use case.

Instructions:
Do steps R6.1 – R6.3 (only R6.3 finds defects).

Step R6.1: Identity the main functionality of a use case and its important system concepts.

Inputs:
1. Use case diagram (UC).

Outputs:
1. System concepts (marked by **blue** on UC).
2. System services provided (marked by **green** on UC).
3. Data necessary to achieve such services (marked by **yellow** on UC).

Instructions: (similar to R5.1 for RD, but here for UC)

Q61.a: Find the unique nouns/concepts in the UC.
Underline and **number consecutively** with a **blue** pen (used in Q61.d).

Q61.b: For each noun, find verbs/actions "to or by" that noun.
Underline and **number in assumed performance order** with a **green** pen.

Q61.c: Mark constraints/conditions in **double-green** (part of service marking).

Q61.d: Also find the information or data to be sent/received in order to perform a certain action. Label the data in **yellow** as "Dx,y", where x,y are the nouns involved.

Step R6.2: Identify and inspect the related sequence diagrams, to identify if the corresponding functionality is described accurately and whether behaviors and data are represented in the right order.

Inputs:
1. Use case diagram (UC), marked-up w/ concepts, services, and data – from R6.1.
2. A set of sequence diagrams (SqD).

Outputs:
1. System concepts (marked in **blue** on SqD).
2. System services (marked in **green** on SqD).
3. Data exchanged between objects (marked in **yellow** on SqD).

Instructions: (cf. above R6.1, but here for SqD)

Q62.a: For each SqD, underline in **blue** the system objects, and with the same noun number (from Q61.a) as in the UC.

Q62.b: Identify the services described in the SqD.
I.e. look at the horizontal message arrows between objects, and possibly **cluster** several arrows into one **servic**e. Underline the identified services in **gree**n,an**dnumber** them in **occurrence order** (top-to-bottom) in the SqD.

Q62.c: Identify information/data exchanged between two system classes (x,y).
Label the data in **yellow** as "Dx,y", as in R6.1.

Step R6.3: Compare the marked-up Use Case / Sequence Diagrams to determine whether they represent the same domain concepts.

Inputs:
1. Use case (UC), w/ marked-up concepts, services, and data – from R6.1.
2. Set of sequence diagrams (SqD), with similar mark ups – from R6.2.

Outputs:
1. Discrepancy reports.

Instructions:

Q63.a: For each **blu**e-marked noun in the UC, search for a similar one in the SqD.
Mark by **blue** star ("*") in the UC if found.
For unstarred nouns in the UC, check also if they possibly are attributes in some class.
The remaining, unstarred nouns from the UC may represent defects, as they are missing in the design (SqD). *[omission*?]

Q63.b: Similarly, for each unmarked noun in the SqD, it may belong to some design-internal or worse: an unused concept. *[extraneous*?]

Q63.c: For each **blu**e-marked service in the SqD, look for the corresponding done in the UC.

E.g. Are SqD classes/objects exchanging messages in the same order as in the UC? If not, this may be a defect.
E.g. Are message parameters on the SqD correctly described in the UC, e.g. right data between right Dx,y etc.?
E.g. Is it possible to "understand" the expected functionality, for instance from data being sent/received, by just reading the SqD?
Report any problem in all this. *[inconsistency?* or *ambiguity*?]

Q63.d: Are **double-gree**n-marked constraints/conditions from the UC being observed by the SqD? *[incorrect fact?*

OORT-7: State Diagram x (Requirement Description / Use Case)

Inputs:
1. The set of all state diagrams (StD).
2. The set of all requirement descriptions (RD).
3. The set of use case diagrams (UC).

Outputs:
1. Discrepancy reports.

Goal: To verify that the state diagrams describe appropriate states of objects and events that trigger state changes as described by the requirements and use cases.

Instructions: For each state diagram(StD) / object, do the steps R7.1 - R7.4:

Step R7.1: Read the state diagram to basically understand what the object it is modeling (nothing more here).

Step R7.2: Read the functional requirements to determine the possible states of the object, which states are adjacent to each other, and which events/actions cause the state changes.

Inputs:
1. Set of requirement descriptions (RD).

Outputs:
1. Object States (marked in **blue** on RD).
2. Adjacency Matrix (AM), recording if there is a state transition from one state to another.

Instructions: (just reading)

Q72.a: Put away the StD and erase any previous stars (“*”) in the RD.
Read through the RD and mark up lightly – with a star (“*”) by a pencil – the places where the actual StD-object/concept is used.

Q72.b: Locate all corresponding places in the RD for all different states of this object, mark these places with a **blue** pen and **number** them from 1..N.

Q72.c: Identify which of the numbered states being the Initial state ("I"), and similarly with the endstate ("E").

Q72.d: Make a N*N Adjacency Matrix (AM) on a **separate sheet of pape**r.
Try to identify possible ij-state transitions here, i.e. if state i can lead to state j.
Put a check mark (“v”) in these AM-ij entries.

Step R7.3: Read the use cases and determine the events that cause state changes.

Inputs:
1. Use case diagrams (UC).
2. Preliminary Adjacency Matrix (AM) – from R7.2.

Outputs:
1. Completed Adjacency Matrix (AM).

Instructions: (just reading)

Q73.a: Read through the use cases and find the ones where the object participates.

Q73.b: For each marked AM-ij entry (i.e. having a transition), document precisely the associated event and/or constraint someplace on the AM paper sheet.

Q73.c: For the blank entries, see if there still might be events that may cause the transition. If not, write a “X” in that entry.

Step R7.4: Read the state diagrams to determine if the described states are consistent with the requirements, and if the transitions are consistent with the requirements and use cases.

Inputs:
1. Marked-up requirement descriptions (RD) – from R7.2.
2. Set of state diagrams (StD).
3. Completed Adjacency Matrix (AM) – from R7.3.

Outputs:
1. Discrepancy reports.

Instructions: Repeat the following steps for each state diagram (StD):

Q74.a: For each **numbered state** in the RD, find the corresponding state in the StD and mark it with a **blue** pen and corresponding number.
Note: state names may be different in the RD and the StD, and overlapping names may not represent identical states.

E.g. Were all states in the RD found in the StD? *[omission*?]
Or maybe some RD states were combined into one StD state, but this was not a sensible combination? *[incorrect fact?]*

E.g. Inversely, were there extra states in the StD? *[extraneous*?]
Or maybe a RD state was split into more than one StD state, but again this was not a sensible combination? *[incorrect fact?]*

Q74.b: Check **transition events and actions** in the AM-matrix/sheet:

E.g. Do all events in the AM appear on the StD? *[omission*?]

E.g. Do all events in the StD appear on the AM? *[extraneous*?]

Q74.c: Check **transition constraints** in the AM-matrix/sheet:

E.g. Do all constraints in the AM appear on the StD? *[omission*?]

E.g. Do all constraints in the StD appear on the AM? *[extraneous*?]

Appendix D – Adjusted OORT

These techniques where developed by Lars Christian Hegde and Tayyaba arif. These where used during our experiment at Ericsson.

OORT-1: Sequence Diagram x Class Diagram
OORT-2: State Diagram x Class Diagram
OORT-3: Sequence Diagram x State Diagram
OORT-4: Class Diagram for internal consistency
OORT-6: Sequence Diagram x Use Case Specification
OORT-7: State Diagram x Use Case Specification

Abbreviations:

Class Diagram – CD
This diagram shows an overview of the classes involved (VOPC – View Of Participating Classes). It also includes the textual description of each class and of the behaviors. This information can be retrieved either by browsing the model using Rational Rose, or by extracting the text using the tool included in Rational Rose, SoDA.

Use Case Specification – UCS
This is a detailed description of the use case described as a basic flow of events and if needed several alternative flows and exceptional flows. Wherever this is stated the Use Case Diagrams are also considered. The specification is not used without the actual use case diagrams.

State Diagram – StD

Sequence Diagram – SqD

NB!

Only the changes in the given diagrams are to be inspected. In the UCS the changes are given in bold, so that they can easily be read. For the other documents, see the additional list of recent changes made to the model.

Also, other referenced documents in the use case specification such as standards may need to be consulted.

In general, you must familiarize yourself with the use case specification before the inspection.

OORT-1: Sequence Diagram x Class Diagram

Inputs:
1. A class diagram (CD including textual description), possibly in several packages.
2. Sequence diagrams (SqD).

Outputs:
1. Annotated versions of above diagrams.
2. Discrepancy reports.

Goal:
To verify that the class diagram for the system describes classes and their relationships in such a way that the behaviours specified in the sequence diagrams are correctly captured.

Instructions:
Do steps R1.1 and R1.2.

Step R1.1:

From a sequence diagram – identify system objects, system services, and conditions.

Inputs:
Sequence diagrams (SqD).

Outputs:
Annotated diagram with:
1. System objects, classes and actors (underlined with **blue** on SqD);
2. System services (underlined with **green** on SqD);
3. Constraints/conditions on the messages/services (circled in **yellow** on SqD).

Instructions: – matches outputs above.

Q11.a: Underline system objects, classes and actors in **blue** on SqD.

Q11.b: Underline system services in **green** on SqD.

Q11.c: Circle constraints/conditions on the messages and services in **yellow** on SqD. This may be restriction on the number of classes/objects to which a message can be sent, restrictions on the global values of an attribute, dependencies between data, or time constraints that can affect the state of the object. Also circle any conditions that determine under what circumstances a message will be sent.

Step R1.2:

Check related class diagrams, to see if all system objects are covered.

Inputs:
1. Annotated sequence diagrams (SqDs) – from R1.1.
2. Class diagrams (CDs).

Outputs:
1. Discrepancy reports.

Instructions: (as questions – here and after)**:**

Q12.a: Can every **object, class and actor** in the SqD be found in the CD? If an actor cannot be found, you need to consider whether the actor must be represented as a class in the system in order to provide necessary behaviour. If the desired behaviour can be achieved without explicit representation, it's not to be considered an inconsistency.
Possible [*inconsistency?*]

Q12.b: Is there a message on the sequence diagram for which the receiving class does not contain an appropriate behaviour on the class diagram? [*inconsistency?*]

Q12.c: Are all system services covered by (low-level) messages in the SqD?
To have an idea of what the system services are, its important to keep the requirements in mind. When the system services do not have appropriate behaviours, it means that no class assumes responsibility for a particular service.
Possible [*omission?*]

Q12.d: Is there an association or other relationship between two classes in case of message
exchanges? If a message is exchanged between two classes they must be related in some way. [*omission?*]

Q12.e: Is there a mismatch in behaviours or in how constraints / conditions are formulated between the two documents? [*inconsistency?*] Are there any missing behaviours, without which the system service cannot be achieved? [*omission?*]

Q12.f: Can the constraints from the SqD in R1.1 be fulfilled in the class diagram?

E.g Is there a limit on the number of objects that can receive a message? The constraint should appear as cardinality information for the appropriate association in the CD.

E.g Is there a specified range of permissible values for data? The constraint should appear as a value range on an attribute in the CD.

E.g Are there dependencies between data or objects? ("a bill object cannot exist without a purchase object")This information should be included as a constraint or relation on the CD.

E.g Is there any timing constraints? This information should be included as a constraint or relation on the CD.

Report any issues [inconsistency?]

Q12.g: **Overall design comments**, based on own experience, domain knowledge, and understanding.

E.g. Do the messages and their parameters make sense for this object?
E.g. Are the stated conditions appropriate?
E.g. Are all necessary attributes defined?
E.g. Do the defined attributes/functions on a class make sense?
E.g. Do the classes/attributes/functions have meaningful names?
E.g. Are class relationships reasonable and of correct type? (ex. association vs. composition relationships).

Report any problems. [*incorrect fact?*]

OORT-2: State Diagram x Class Diagram

Inputs:
1. A set of class diagram (including textual description).
2. A set of state diagrams for the system objects.

Outputs:
1. Discrepancy reports.

Goal:
To verify that the classes are defined, so that they can capture the functionality specified by the state diagram.

Instructions:
Repeat steps R2.1 – R2.3 for each state diagram (StD).

Step R2.1:

Read state diagram to understand possible states and their transition.

Inputs:
1. State diagram (StD).

Outputs:
1. Object states (underlined in **blue** on StD)
2. Transition actions/conditions (underlined in **green** on StD)
   I.e., a marked-up state diagram is produced, used in R2.2 and R2.3.
3. Discrepancy reports.

Instructions:

Q21.a: Identify which class this diagram is modelling. [*omission?*]

Begin at the filled circle and follow the transitions until you reach an end state (double circle). Make sure you cover all states and transitions.

Q21.b: Underline the name of each object state (by **blue** pen).

Q21.c: Underline the transition actions (represented by arrows) and conditions (by **green** pen).

Q21.d: Can you understand what's going on with the object from Q21.b-c above? [*ambiguity?*]

Step R2.2:

Identify the state diagram's associated class or class hierarchy, attributes and behaviour.

Inputs: partly from state diagram (StD)
1. Set of class diagram (CD).
2. Object states (underlined in **blue** on StD – from R2.1)
3. Transition actions/conditions (underlined in **green** on StD – from R2.1)

Outputs:
1. Discrepancy reports.

Instructions:

Q22.a: In the CD, identify the class or class hierarchy that corresponds to this StD. Did you find the corresponding class? If not, you have found a defect. [inconsistency*?*]

Q22.b: Find out how a **blue-**underlined state is represented, i.e. has the class captured each modelled state in a unique way?

E.g. By an explicit attribute. An attribute exists with possible values that correspond to system states.

E.g. By an implicit attribute. An object state depends on the value of an attribute, but the state is not recorded explicitly.

E.g. By a combination of attributes.

E.g. By subtyping of the actual object (consult the class hierarchy). E.g subclasses " fixed_rate_loan" and "variable_rate_loan" can be considered states of parent class "loan".

Mark each blue-underlined state with a star (*) in the StD as they are found. Does the StD capture all the states? The object cannot capture certain states (inconsistency), or it is not clear how they are represented (ambiguity)?
Report the result. [*inconsistency?* or *ambiguity?*]

Q22.c: Are all **green-**underlined transition actions/conditions covered by class behaviour?

Make sure to look through the whole class hierarchy for the current class when looking for behaviours. Starr (*) each green highlighted transition action in StD as corresponding behaviour is found in the CDe
If not: error. [*inconsistency?*]

Q22.d: Could you identify the object data used to verify the green marked transition conditions? Are they consistent between the state diagram and the class description?
If not: error. [*inconsistency?*]

Step R2.3:

Check Class diagram to see if the states are appropriate.

Inputs: from state diagram (StD)

1. Object states (underlined in **blue** on StD from R2.1)
2. Transition actions and conditions (underlined in **green** on StD from R2.1)

Outputs:

1. Discrepancy reports.

Instructions:

Q23.a: From your domain knowledge, are all relevant states for the behavior of the class defined in the StD? [*incorrect fact?*]

Q23.b: For each unmarked state in the StD, consider if the state is really needed or if it represents extraneous information. [*incorrect fact?* or *extraneous?*]

Q23.c: For each unmarked transition action/condition in the StD, report inconsistency between CD and StD. [inconsistency?]

OORT-3: Sequence Diagram x State Diagram

Inputs:
1. A set of state diagrams for several objects.
2. A set of sequence diagrams.

Outputs:
1. Discrepancy reports.

Goal:
To verify that every state transition for an object can be achieved by the messages sent and received by that object.

Instructions:
Repeat steps R3.1 – R3.3 for each state diagram (StD).

Step R3.1:

Read the state diagram to understand the possible states of the object and the actions that trigger transition between them.

Inputs:
1. Given state diagram (StD).

Outputs:
1. State diagram (StD), with transition actions labelled and highlighted in **green**.
2. Discrepancy reports.

Instructions:

Q31.a: Determine which class is being modelled. Missing? [*omission?*]

Q31.b: Trace the sequence of states (from start state to end state) and the transition actions through the state diagram. Highlight transition actions (represented by arrows) as you come to them using a **green** pen and give each action a unique label (A1, A2…). Transitions with equal names and end state is considered equal, and can be labeled with the same label.

Q31.c: In general, do these transitions/actions and states make sense and are they understandable for such an object? Is it possible to understand and describe what is going on with the object just by reading this state machine? [*ambiguity?*]

Step R3.2:

Read the sequence diagrams to understand how the transition actions are achieved by messages sent to/from the relevant object.

Inputs:
1. State diagrams (StD), with transition actions labelled and highlighted in **green** (from R3.1).
2. Set of sequence diagrams (SqD).

Outputs:
1. Annotated sequence diagrams (SqD), with matching object messages starred and labeled in **green**.
2. Discrepancy reports.

Instructions:

Q32.a: Take the sequence diagrams and choose the ones that use the object modelled by the state diagram, use only this subset for the remainder of this step. Could you find the sequence diagrams in which the object participates? [*omission?* or *extraneous?*]

For each relevant sequence diagram (SqD) do below points Q32.b-e:

Q32.b: Read the diagram to identify the system service being described and the messages that this object receives.

Q32.c: Identify the object states in the StD, being semantically related to the actual system service.

Q32.d: Map the object messages on the SqD to the state transitions on the StD. Each transition action may map to one message, or a sequence of messages. To accomplish this, consider the semantics of the messages. Were there additional messages needed to achieve the state transition? Mark the related SqD-messages and StD-transitions actions, both with a **green** star (*). Label the SqD-messages with the same label given to their associated StD-action (highlighted and labelled arrows from Q31.b).

Q32.e: Look for constraints and conditions on the above SqD-messages. Check if the same constraint/condition information stands in both diagrams. [*inconsistency?*] Look to see that any constraints/conditions found are captured somehow on the state diagram. If there is some information not captured on the state diagram, you need to consider if it is important enough to the object states that it should have been represented somehow? Such SqD-information may be correspondingly expressed in the StD by:

1) State information	(e.g. t>0)
2) Transition information	(what occurs when t>0?)
3) Nothing	(not relevant for StD)

Step R3.3:

Review the marked-up diagrams to make sure that all transition actions are accounted for.

Inputs:
1. Transaction actions on the given StD (annotated in **green –** from R3.1)
2. Object messages on the SqD (annotated in **green –** from R3.2)

Outputs:
1. Discrepancy reports.

Instructions:

Q33.a: Look for unstarred **transition actions** in the StD that could not be associated with object messages. Look for constraints or events belonging to an unstarred transition action, are these somehow represented by a message or sequence of messages or an event performed by an actor? If not, report the unstarred transition actions (cf. Q32.d). Report these. [*inconsistency?*]

Q33.b: Are the **event order** the same in the StD and SqD, i.e. check if with starred messages and transition actions in the SqD appear in logical order? [*inconsistency?*] E.g. that action Ax on a later transition in the StD actually occurs after an action Ay on an earlier transition.

OORT-4: Class Diagram for internal consistency

Inputs:
1. A class diagram (CD including textual description), possibly in several packages.

Outputs:
1. Discrepancy reports.

Goal:
To verify that the detailed descriptions of classes contain all the information necessary, and that the description of classes make semantic sense.

Instructions:
Repeat steps R4.1 and R4.2 for each class in the class diagram (CD).

Step R4.1:

Read the class diagram to understand the necessary properties.

Inputs:
1. Given class from class diagram (CD).

Outputs:
1. Discrepancy reports.

Instructions:

Q41.a: Is there a textual description for each class? [*omission?*]

Q41.b: Is the name and textual description of this class meaningful? [*ambiguity?*]

Q41.c: Can this class meaningfully contain all these attributes and with given types? [*ambiguity?* or *incorrect fact?*]

Q41.d: On **behaviour and constraints**:

E.g.	Are behaviours described at the same level of detail? [*inconsistency?*]
E.g.	In general, should this class really contain all these behaviours and constraints? [*incorrect fact?*]
E.g.	Do the behaviours and constraints use available behaviours or attributes from elsewhere, and are they defined? [*omission?* or *ambiguity?*]
E.g.	Do the behaviours and constraints rely "excessively" on attributes in remote classes? I.e. too high coupling? [*miscellaneous?*]

Q41.e: In case of use of **inheritance** in the CD:

E.g. In general, is it "meaningful" for the given class be a supertype/subtype of the given subclasses/superclass? [*miscellaneous?*]

Q41.f: Check that all **relationships** are correctly described:

E.g. Do they have the right cardinalities? [*inconsistency?*]

E.g. In general, do the stated relationships "make sense", such as composition vs.

aggregation vs. association vs. inheritance etc.? [*miscellaneous?*]

E.g. Is an attribute used to represent a relationship, and does this have the right type (a reference or sets of references)? [*inconsistency?*]

OORT-6: Sequence Diagram x Use Case Specification

Inputs:
1. A use case specification (UCS) for a part of the system, with its services.
2. One or more sequence diagrams (SqD) for relevant system objects and services.

Outputs:
1. Discrepancy reports.

Goal:
To verify that sequence diagrams describe an appropriate combination of objects and messages that capture the functionality from the use case specification.

Instructions:
Do steps R6.1 – R6.3 (only R6.3 finds defects).

Step R6.1:

Identify the main functionality of a use case specification and its important system concepts.

Inputs:
1. Use case diagram (UC).

Outputs:
1. System concepts (annotated by **blue** on UCS).
2. System services provided (annotated by **green** on UCS).
3. Data necessary to achieve such services (annotated by **yellow** on UCS).

Instructions:

Q61.a: Find the unique nouns/concepts in the UCS.
Underline and **number consecutively** with a **blue** pen.

Q61.b: For each noun, find verbs/actions "to or by" that noun.
Underline and **number in assumed performance order** with a **green** pen.

Q61.c: Mark constraints/conditions in **double-green** (part of service marking).
Constraints and conditions are given in plain text in the UCS. Read it carefully to identify the constraints/ conditions that are necessary for the marked actions to be performed.

Step R6.2:

Identify and inspect the related sequence diagrams, to identify if the corresponding functionality is described accurately and whether behaviours and data are represented in the right order.

Inputs:
1. Use case specification (UCS), annotated w/ concepts, services, and data – from R6.1.
2. A set of sequence diagrams (SqD).

Outputs:
1. System concepts (annotated in **blue** on SqD).
2. System services (annotated in **green** on SqD).
3. Data exchanged between objects (annotated in **yellow** on SqD).

Instructions: (cf. above R6.1, but here for SqD)

Q62.a: For each SqD, underline in **blue** the system objects, and with the same noun number (from Q61.a) as in the UCS.

Q62.b: Identify the services described in the SqD. I.e. look at the horizontal message arrows between objects, and identify the **services** described in the SqD. To do this, look at the information exchanged (horizontal message arrows) between objects in the SqD. If the messages are very detailed, you may need to **cluster** several messages together to identify the service they provide. Underline the identified services in **green**, and **number** them in **occurrence order** (top-to-bottom) in the SqD.

Step R6.3:

Compare the marked-up Use Case Specification/ Sequence Diagrams to determine whether they represent the same domain concepts.

Inputs:
1. Use case specification (UCS), w/ annotated concepts, services, and data – from R6.1.
2. Set of sequence diagrams (SqD), with similar annotation – from R6.2.

Outputs:
1. Discrepancy reports.

Instructions:

Q63.a: For each **blue**-marked noun in the UCS, search for a similar one in the SqD. Mark by **blue** star ("*") in the UCS if found.
For unstarrred nouns in the UCS, check also if they possibly are attributes in some class. Check the CDe. The remaining, unstarred nouns from the UCS may represent defects, as they are missing in the design (SqD). [*omission?*]

Q63.b: Similarly, for each unmarked noun in the SqD, it may belong to some design-internal or worse: an unused concept. Mark the **nouns** on the sequence diagram with a

blue cross (x) if they appear only on the SqD. Are there any crossed nouns in the SqD? This may be an extraneous concept or a lower-level concept. Try do decide whether this concept is necessary for the high-level design or not. Report this. [*extraneous?*]

Q63.c: For each **green**-marked service in the SqD, look for the corresponding done in the UCS.

E.g. Are SqD classes/objects exchanging messages in the same order as in the UCS? If not, this may be a defect.

E.g. Is it possible to "understand" the expected functionality, for instance from data being sent/received, by just reading the SqD?

Report any problem in all this. [*inconsistency?* or *ambiguity?*]

Q63.d: Are all **double-green** marked constraints/conditions from the UCS observed in the SqD? Are all behaviour and data directly concerned with the UCS? [*incorrect fact?]*

OORT-7: State Diagram x Use Case Specification

Inputs:
1. The set of all state diagrams (StD).
2. The set of use case specification (UCS)

Outputs:
1. Discrepancy reports.

Goal:
To verify that the state diagrams describe appropriate states of objects and events that trigger state changes as described by the Use Case Specification.

Instructions:
For each state diagram (StD) / object, do the steps R7.1 - R7.3:

Step R7.1:

Read the state diagram to basically understand what the object it is modelling (nothing more here).

Step R7.2:

Read the use case specification (UCS) to determine the possible states of the object, which states are adjacent to each other, and which events/actions cause the state changes.

Inputs:
1. Set of requirement specification (UCS).

Outputs:
1. Object States (marked in **blue** on UCS).

Instructions: (just reading)

Q72.a: Put away the StD and erase any previous stars (“*”) in the UCS.
Read through the UCS and mark up lightly – with a star (“*”) by a pencil – the places where the actual StD-object/concept is used.

Q72.b: Locate all corresponding places in the UCS for all different states of this object, mark these places with a **blue** pen and **number** them from 1..N.

Q72.c: Identify which of the numbered states being the Initial state ("I"), and similarly with the end state ("E").

Step R7.3:

Read the state diagrams to determine if the described states are consistent with the specifications, and if the transitions are consistent with use case specification.

Inputs:
1. Annotated use case specification (UCS) – from R7.2.
2. Set of state diagrams (StD).

Outputs:
1. Discrepancy reports.

Instructions: Repeat the following steps for each state diagram (StD):

Q73.a: For each **numbered state** in the UCS, find the corresponding state in the StD and mark it with a **blue** pen and corresponding number. Note: state names may be different in the UCS and the StD, and overlapping names may not represent identical states.

E.g. Were all states in the UCS found in the StD? [*omission?*] Or maybe some UCS states were combined into one StD state, but this was not a sensible combination? [*incorrect fact?]*

E.g. Inversely, were there extra states in the StD? [*extraneous?*] Or maybe a UCS state was split into more than one StD state, but again this was not a sensible combination? [*incorrect fact?]*

Q73.b: Check if the state **transition events and actions** correspond with the transition events and actions in the use case specification.
[*omission?*] [*extraneous?*]

Q73.c: Check if the **transition constraints** correspond with the transitions constraints in the use case specification.
[*omission?*] [*extraneous?*]

Appendix E – Questionnaire

Experience Questionnaire (for OO Reading techniques)

Name: _____

Software Development Experience

Please rate your experience in this section with respect to the following 5-point scale:

1 = none
2 = studied in class or from book
3 = practiced in a class project
4 = used on one project in industry
5 = used on multiple projects in industry

Experience with Requirements

?	Experience writing requirements	1	2	3	4	5
?	Experience writing use cases	1	2	3	4	5
?	Experience reviewing requirements	1	2	3	4	5
?	Experience reviewing use cases	1	2	3	4	5
?	Experience changing requirements for maintenance	1	2	3	4	5

Experience in Design

?	Experience in design of systems	1	2	3	4	5
?	Experience in design of systems from requirements / use cases	1	2	3	4	5
?	Experience with creating Object-Oriented (OO) designs	1	2	3	4	5
?	Experience with reading OO designs	1	2	3	4	5
?	Experience with the Unified Modeling Language (UML)	1	2	3	4	5
?	Experience changing designs for maintenance	1	2	3	4	5

Experience in Coding

?	Experience in coding, based on requirements / use cases	1	2	3	4	5
?	Experience in coding, based on design	1	2	3	4	5
?	Experience in coding, based on OO design	1	2	3	4	5
?	Experience in maintenance of code	1	2	3	4	5

Experience in Testing

?	Experience in testing software	1	2	3	4	5
?	Experience in testing, based on requirements / use cases	1	2	3	4	5
?	Experience with equivalence-partition testing	1	2	3	4	5

Other Experience

?	Experience with software project management?	1	2	3	4	5
?	Experience with User Interface (UI) design?	1	2	3	4	5
?	Experience with software inspections?	1	2	3	4	5

Confidential

Because of the material included in this Appendix cannot be published; it was removed from the thesis. The results from the questionnaire can be made available on request and approval from Ericsson.

Appendix F – Experiment Data

The data gathered from the experiment is presented here.

Confidential

Because of the material included in this Appendix cannot be published; it was removed from the thesis. The results from the experiment can be made available on request and approval from Ericsson.