National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de Technologie de l'information

ERB-1069

NRC·CNRC

An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents

Oliver Laitenberger, Colin Atkinson,
Maud Schlich, and Khaled El-Emam
December 1999

Canada

NRC 43614

National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de Technologie de l'information

An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents

Oliver Laitenberger, Colin Atkinson,
Maud Schlich, and Khaled El-Emam

December 1999

An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents

Oliver Laitenberger, Colin Atkinson, Maud Schlich
Fraunhofer Institute for
Experimental Software Engineering
Sauerwiesen 6
D-67661 Kaiserslautern
Germany
{laiten,atkinson, schlich} @iese.fhg.de

Khaled El Emam
National Research Council, Canada
Institute for Information Technology
Building M-50, Montreal Road
Ottawa, Ontario
Canada K1A OR6
Khaled.El-Emam@iit.nrc.ca

Abstract

The basic motivation for software inspections is to detect and remove defects before they propagate to subsequent development phases where their detection and removal becomes more expensive. To maximize this potential, the examination of the artefact under inspection must be as thorough and detailed as possible. This implies the need for systematic reading techniques that tell inspection participants what to look for and, more importantly, how to scrutinise a software document. Recent research efforts have investigated the benefits of scenario-based reading techniques for defect detection in functional requirements and functional code documents. A major finding has been that these techniques help inspection teams find more defects than existing state-of-the-practice approaches, such as, ad-hoc or checklist-based reading (CBR). In this paper we describe and experimentally compare one scenario-based reading technique, namely perspective-based reading (PBR), for defect detection in object-oriented design documents using the notation of the Unified Modelling Language (UML) to the more traditional CBR approach. The comparison was performed in a controlled experiment with 18 practitioners as subjects. Our results indicate that PBR teams discovered, on average, 58% of the defects and had an average cost per defect ratio of 56 minutes per defect. In this way, PBR is more effective than CBR (i.e., it resulted in inspection teams detecting, on average, 41% more unique defects than CBR). Moreover the cost of defect detection using PBR is significantly lower than CBR (i.e., PBR exhibits, on average, a 58% cost per defect improvement over CBR). This study therefore provides evidence demonstrating the efficacy of PBR scenarios for defect detection in UML design documents. In addition, it demonstrates that a PBR inspection is a promising approach for improving the quality of models developed using the UML notation.

Keywords: Software Inspection, Perspective-based Reading, Controlled Experiment, Object-Orientation, Unified Modeling Language, Fusion

1 Introduction

Since Fagan's initial work presented in 1976 (Fagan, 1976) software inspection has emerged in software engineering as one of the most effective and efficient methods for software quality improvement. It has been claimed that inspections can lead to the detection and correction of anywhere between 50 percent and 90 percent of the defects in a software document (Fagan, 1986; Gilb and Graham, 1993). Moreover, since inspections can be performed at the end of each development phase and since the defects are typically found close to the point where they are introduced, rework costs[1] can be reduced considerably. For example, a Monte Carlo simulation using parameters collected from the literature has indicated that, on average, the implementation of code inspections reduces life cycle defect detection costs by 39

[1] This constitutes the costs associated with correcting defects.

percent, and that the implementation of design inspection reduces life cycle defect detection costs by 44 percent (Briand et al., 1998)[2].

A software inspection usually consists of several activities such as planning, defect detection, defect collection, and defect correction (Laitenberger and DeBaud, 2000)[3]. Inspection planning is performed by an organizer who schedules all subsequent inspection activities. The defect detection activity can be performed either by inspectors individually or in a group meeting. Recent empirical findings reveal that the synergy effect of inspection meetings is rather low in terms of impact on defects detected (Johnson and Tjahjono, 1998; Land et al., 1997; Votta, 1993). Therefore, defect detection can be regarded as an individual rather than a group activity. Defect collection, on the other hand, is often performed in a team meeting (i.e., an inspection meeting) led by an inspection moderator. The main goals of the team meeting are to consolidate the defects inspectors have detected individually, to eliminate false positives, and to document the real defects. An inspection often ends with the correction of the detected defects by the author.

Although each of these activities is important for a successful inspection, the key part of an inspection is the defect detection activity. Throughout this activity inspectors read software documents and check whether they satisfy quality requirements, such as correctness, consistency, testability, or maintainability. Each deviation is considered a defect. Because of its importance, adequate support for inspectors during defect detection can potentially result in dramatic improvements in inspection effectiveness and efficiency. The particular type of support that we focus on in this paper is the reading technique that is used during the defect detection activity[4].

In practice, most industrial inspection implementations use ad-hoc or checklist-based reading (CBR) during defect detection (Fagan, 1976; Gilb and Graham, 1993; Laitenberger and DeBaud, 2000). Ad-hoc reading, as its name implies, provides no explicit advice for inspectors as to how to proceed, or what specifically to look for during the reading activity. Hence, inspectors must resort to their own intuition and experience to determine how to go about finding defects in a software document. Checklists offer stronger support mainly in the form of yes/no-questions that inspectors have to answer while reading a software document. Gilb and Grahams' manuscript on software inspection states that checklist questions interpret specified rules within a project or an organization (Gilb and Graham, 1993). Such rules may be, for example, development standards.

Although the checklist-based approach offers more reading support than ad-hoc, it has four principal shortcomings. The first stems from the fact that a checklist is often based upon past defect information (Chernak, 1996). If no such information is available, the checklist questions are often taken from the literature, which is a kind of reuse of defect experience of the organization that created the checklist. In both cases, inspectors do not pay attention to defects and defect types that were not previously detected and, therefore, may miss some defects or whole classes of defects. Second, a checklist often contains too many questions, and it is rare to find concrete instructions on how to answer a particular one. Therefore, it is often unclear for an inspector, when and based on what information, he or she is to answer the questions. The lack of concrete guidance for answering the questions is linked with the third shortcoming: a checklist does not require an inspector to document his or her analysis. Thus, the result of the analysis effort is not repeatable by others and depends heavily on the individual inspector. Finally, a checklist requires each inspector to check all information in the provided documents for possible defects, which may cause him or her to get swamped with many unnecessary details (Parnas and Weiss, 1987).

A more recent approach, scenario-based reading (Basili, 1997), has been proposed to tackle some of these deficiencies. The basic idea of scenario-based reading techniques is the use of so-called scenarios that describe *how* to go about finding the required information in a software document, as well as *what* that information should look like.

---

2 Both of these figures assume that the defect detection life cycle prior to the introduction of inspections consisted only of testing activities.

3 In this article, we model the inspection process in terms of its main activities. This allows us to be independent of a specific inspection implementation, such as that of Fagan (Fagan, 1976) or of Gilb and Graham (Gilb and Graham, 1993).

4 Other types of support can also be effective. For example, training sessions in program comprehension as presented in (Rifkin and Deimel, 1994) can be beneficial to maximize the number of detected defects in code inspection.

To date, reading techniques and software inspections in general, have been used primarily in connection with textual documents resulting from conventional structured development processes, such as functional requirements documents or functional code modules (Basili et al., 1996; Laitenberger et al, 2000, Porter et al., 1995; Porter and Votta, 1998). Object-oriented models, particularly of the graphical form, have so far not been adequately addressed by inspection methods. This represents a problem for two reasons. First, over the past decade object-oriented development methods have replaced conventional structured methods as the embodiment of "goodness" in software development, and are now the approach of choice in most new software development projects. The recent publication of the Unified Modelling Language (UML) notation (Booch et al., 1999) accentuates this trend. Reading techniques that are limited to conventional structured methods, therefore, are expected to become less and less relevant for the development of new software products as these methods are superseded. Second, despite its many beneficial features, low defect density is not one of the strong points of the object-oriented paradigm. On the contrary, some empirical studies have shown that object-oriented software is more error-prone than functional software (Hatton, 1998). Object-oriented methods would therefore benefit enormously from the availability of advanced, experimentally-validated defect detection techniques. To our knowledge, the question of how to inspect object-oriented analysis and design documents written in the UML notation has not been subject to controlled or quasi-controlled experimental work.

In this paper we focus on one particular scenario-based reading technique, namely perspective-based reading (PBR). After describing the details of the PBR technique for defect detection in object-oriented UML design documents, we present a controlled experiment with 18 practitioners as subjects. During the experiment, the effectiveness[5] of PBR and its cost per defect ratio are compared with those of CBR.

Briefly, our results indicate that inspection teams applying PBR discovered, on average, 58 percent of the defects in a software document. Moreover, PBR teams exhibit an average cost per defect ratio of 56 minutes per defect. In this way, inspection teams using PBR for defect detection have a higher effectiveness than CBR (an improvement of 41%), as well as a lower cost per defect ratio than those applying CBR (a reduction of 58%). The remainder of this paper is organised as follows. Section 2 describes in more detail the investigated reading techniques, the research questions, and the experimental hypotheses. Section 3 presents the controlled experiment. This includes a discussion of the experimental design, a description of the environment and the subjects, the dependent and independent variables, how the studies were conducted, and the data analysis methods. Section 4 follows with a presentation of the experimental results and their interpretation. Section 5 discusses the threats to internal and external validity. Finally, Section 6 concludes with a summary and directions for future work.

2 Background

This section presents an overview of existing reading techniques for software inspections, a brief introduction to the Unified Modeling Language (UML), and the hypotheses that we test in our experiment.

2.1 Reading Techniques for Defect Detection in Software Inspection

In addition to ad-hoc and CBR, a number of other reading techniques have been proposed in the literature. These are briefly reviewed below.

Reading by Stepwise Abstraction is a technique that requires a more rigorous examination of the software artefact than either ad-hoc or Checklists (Dyer, 1992; Linger et al., 1979). Its use has often been described in the context of the Cleanroom Software Development Method because Cleanroom offers a set of formally described development artefacts that are particularly suited for this reading technique.

S*cenario-based reading techniques* suggested by Basili (Basili, 1997) extend the work of Parnas and Weiss on *Active Design Reviews* (Parnas and Weiss, 1987) and allocate specific responsibilities to inspectors. In addition, scenario-based reading techniques provide guidance for inspectors, in the form of so-called scenarios, as to what to check and how to perform the required checks. A scenario typically

[5] In this study effectiveness is defined as the proportion of all defects in the design that were found by applying one reading technique.

consists of a limited, specific set of questions and a detailed set of instructions for an inspector on how to perform the checking.

To date, researchers have suggested three different scenario-based reading techniques for defect detection in documents developed according to functional development processes: *Defect-based Reading* (Porter et al., 1995), a scenario-based reading technique based on function points (Cheng and Jeffrey, 1996), and *Perspective-based Reading* (Basili et al., 1996).

Porter et al. (Porter et al., 1995) describe reading scenarios based on a defect taxonomy for the inspection of functional requirements documents. These scenarios are derived from defect classes and consist of a specific set of questions an inspector has to answer while reading a requirements document. The scenario questions focus on the detection of defects belonging to a particular defect class. Some experiments with students as subjects found that subjects using the defect-based reading technique detect more defects in requirements documents than subjects applying either ad-hoc or CBR (Porter et al., 1995; Miller et al., 1998). Similar results were found with professional subjects (Porter and Votta, 1998).[6]

Cheng and Jeffery (Cheng and Jeffery, 1996) base the development of scenarios on Function Point Analysis (FPA). FPA defines a software system in terms of specific function point elements, such as inputs, files, inquiries, and outputs. A function point scenario consists of questions about a specific function point element. The researchers carried out an experiment with students as subjects to compare the function point scenario approach to ad-hoc reading for defect detection in requirements documents. The experimental results show that, on average, subjects following the ad-hoc approach found more defects than subjects following the function-point scenarios. However, the authors assert that experience was a confounding factor that biased the results of this experiment.

Basili et al. (Basili et al., 1996) present perspective-based reading scenarios. A perspective-based reading scenario supports the checking of a document from a particular stakeholder's perspective. In a way, the PBR technique synthesises ideas that have already appeared in previous articles on software inspection, but have never been worked out in detail. For example, Fagan (Fagan, 1976) reports that a piece of code should be inspected by its real tester, while Fowler (Fowler, 1986) suggests that each inspection participant should take a particular point of view when examining the work product. Finally, Graden et al. (Graden et al., 1986) state that inspectors must identify any required functional responsibility they assume in inspecting a document. That is, each inspector must denote the perspective (customer, requirements, design, test, maintenance) from which they have evaluated a document.

A perspective-based reading scenario consists of activities an inspector is to perform to extract information or use the information for specific purposes, and questions to analyse the extracted information. The benefits of the PBR technique have been demonstrated several times. In the context of a controlled experiment at NASA (Basili et al., 1996), Basili et al. compared the PBR approach to a specific NASA reading approach, which evolved over several years. They found that for some requirements documents, individual subjects using the PBR technique were more effective at defect detection in requirements documents than subjects using the NASA reading approach. Moreover, they simulated team meetings and found the PBR approach superior to the NASA approach. In a quasi-experiment, Laitenberger et al. (Laitenberger et al., 2000) demonstrated that PBR increases inspection effectiveness and decreases the cost per defect ratio compared to CBR when used for defect detection in functional C-code. The quasi-experiment was performed with 60 professional developers in an industrial context. Furthermore, Laitenberger et al. (Laitenberger et al., 2000) examined the theoretical mechanism for the PBR benefits. They demonstrated that because PBR focuses the inspectors' attention on different subsets of defects in a document, this leads to greater inspection effectiveness. These improvements in effectiveness result from the fact that the probability of finding a particular defect through PBR is greater than for CBR.

[6] Two other student experiments did not confirm the superior defect detection capability of defect-based reading (Fusaro and Lanubile, 1997; Sandahl et al., 1998). However, for these experiments the authors noted that the subjects lacked domain experience and "had to learn too many new things" at the same time, and this may have had an impact on the results. A meta-analysis of the five cited DBR experiments (Hays, 1999) did find that the combined p-value was less then 0.05 and that the combined effect size was in the direction favouring DBR. However, the estimated standard deviation for this effect size was quite large.

The review of the literature above indicates that there is initial empirical evidence demonstrating the utility of defect-based reading and perspective-based reading. However, this evidence has been limited to documents developed according to functional development approaches.

Some work has recently been reported in the area of object-oriented inspections. Travassos et. al (Travassos et al., 1999) have suggested a scenario-based reading technique for defect detection in object-oriented design documents called "traceability-based reading." The scenarios describe how to perform correctness and consistency checks among various UML-models. The researchers also performed a case study (with no experimental control) to show the feasibility of this approach. However, further corroborative evidence from a controlled study is so far unavailable. Moreover, it remains unclear why this approach is expected to be more cost-effective than the ad-hoc or CBR approach, how to tailor this technique to other UML models, and how to integrate it into software inspection processes. Therefore, this reading technique remains at the early development stages.

Laitenberger and Atkinson (Laitenberger and Atkinson, 1999) describe a PBR-based approach for defect detection in object-oriented development models. They provide detailed guidelines for tailoring the technique to different development phases and integrating it into existing inspection processes. These guidelines, together with a theoretical explanation as to why PBR is more cost-effective than CBR(Laitenberger et al., 2000), provides a good platform for performing a systematic comparison of the effectiveness of PBR and CBR techniques in the inspection of object-oriented UML-based design documents. This is the approach applied in the experiment reported in this paper.

2.2 Overview of the UML

Since its publication by the Object Management Group (OMG) in 1997, the UML has been widely adopted within the software industry, and has become the defacto modelling notation for object-oriented analysis and design. Although primarily a unification of the OMT (Rumbaugh et al., 1991) , Booch (Booch et al., 1999), and Objectory methods (Jacobson, 1992), the UML has also received input from numerous other sources. Figure 1 lists the eight distinct diagram types defined in the UML.

USE CASE DIAGRAMS	- system usage patterns
STATIC STRUCTURE DIAGRAMS	- static relationships of classes/objects
BEHAVIOR DIAGRAMS	
STATE DIAGRAMS	- state transitions for individual classes
ACTIVITY DIAGRAMS	- flow charts, work flow
SEQUENCE DIAGRAMS	- interactions (temporal emphasis)
COLLABORATION DIAGRAMS	- interactions (object emphasis)
IMPLEMENTATION DIAGRAMS	
COMPONENT DIAGRAMS	- static software organization
DEPLOYMENT DIAGRAMS	- process to processor mappings

Figure 1: UML Diagram Types.

Use-case diagrams are based largely on those of the Objectory method (Jacobson, 1992). The purpose of this type of diagram is to provide an overview of the high-level services to be supported by the system, the relationships among them, and the actors involved. The use-case diagram is normally accompanied by a textual description of the use cases.

Static structure diagrams serve to describe the classes and/or objects in the system, and the primary relationships between them. In general, a static structure diagram can contain both classes and objects. Diagrams that contain mainly the former are termed *class diagrams*, while those that contain only the latter are called *object* diagrams. Class diagrams are by far the most predominant form of static structure diagram.

State diagrams, *activity diagrams*, *sequence diagrams,* and *collaboration diagrams* are collectively known as *behavior diagrams,* since they describe the behavior of an individual object, or a group of objects. In fact, two of these diagram types capture identical information, but from a different perspective. Sequence diagrams show a sequence of inter-object messages from a chronological, time-ordered perspective, whereas collaboration diagrams capture the same information from a structural, object-oriented perspective. In contrast, *state diagrams* capture the external behavior of an object in terms of abstract states, events, and transitions between states. *Activity diagrams* are a special case of state diagrams that only have *activity states*: states with internal activities and transitions corresponding to the completion of those activities. In practice, they can be used in the style of both the classic flow chart and data flow diagram.

Implementation diagrams are concerned with the grouping and distributing of software elements among available processing units. *Component diagrams* describe how objects are grouped into independently deployable software elements, and *deployment diagrams* describe how they are actually deployed on the available hardware.

In our experiment we primarily deal with use-case, sequence, class, and collaboration diagrams.

2.3 Experimental Hypotheses

The fundamental rationale underlying software inspection is to detect defects in a software document before these defects propagate to subsequent development phases where their detection costs escalate. Furthermore, the cheaper it is to find defects during inspections, the greater the cost savings from the implementation of inspections (Briand et al., 1998). It is therefore important to maximize the number of defects detected through inspections, and to minimize the costs of detecting those defects.

We focus our evaluations on two important aspects of software inspections in this paper: their effectiveness and their cost[7]. Effectiveness is defined as the proportion of defects in the document that were found during an inspection. Cost is defined in terms of the effort involved in finding a single defect. Effort is the most important factor in determining the cost of a software inspection. We also consider the cost per defect found for two phases of an inspection separately: defect detection and defect collection.

We consider in this experiment the team results as our unit of analysis for the following reason. When using CBR, individual subjects do not adopt a particular perspective while reading the documents, whereas they do when they are implementing PBR. With a perspective, a subset of the defects in the document has a high probability of being detected, while the remainder of the defects has a relatively low probability of being detected by that perspective (Laitenberger et al., 2000). Conversely, with CBR one would expect more uniformity in the probability of detection across defects. This reasoning makes it clear that we do not necessarily expect individual PBR inspectors to be more cost-effective than individual CBR inspectors. Rather, we expect the benefits of PBR to become apparent at the team level. In fact, it is the team result that determines the results of an inspection. One may argue that individual variability influences the team results. However, in the context of our experiment, individual variability is controlled by random assignment of subjects to teams. We can therefore state the following expectations for the experiment:

1. **The Effectiveness of PBR is Larger than the Effectiveness of CBR for Teams.**
   We expect that inspection teams detect more defects using PBR than CBR. This derives from the fact that the probability of finding a unique defect through PBR is greater than for CBR (Laitenberger et al., 2000).
2. **Per-defect Detection Cost for Teams is Lower with PBR than with CBR.**
   The PBR technique requires an inspector to perform certain activities based on the content of the inspected document and to actively work with the document instead of passively scanning through it. It is therefore expected that an individual inspector will spend at least as much effort for defect

[7] Another evaluative criterion of software inspections is their interval (i.e., calendar time elapsed) (Votta, 1993). However, this is not addressed in the current study.

detection using PBR than CBR. The effect that we anticipate states that the increase in detected defects must be larger than the additional effort spent for defect detection using PBR. In this case, the total cost to detect a defect using CBR will be higher than for PBR.

3. **Meeting Cost for Teams is Lower with PBR than with CBR.**
   Performing these extra activities during PBR is expected to result in a better understanding of the document. Therefore, during the meeting, inspectors do not have to spend a lot of extra effort in explaining the defects that they found to their colleagues in the team. Furthermore, it will take less effort to resolve false positives due to the enhanced understanding. The better understanding is expected to translate to an overall reduction in meeting effort for PBR than for CBR:[8] Therefore, one would hypothesise that the meeting cost for PBR will be less than for CBR.

4. **Overall Inspection Cost is Lower with PBR than with CBR.**
   Following from the above two arguments about the cost per defect for defect detection and for the meeting, we would expect that the overall cost per defect for both phases to be smaller for PBR than for CBR.

Based on the above explanation of the expected effects and the expectations, below we state our four null hypotheses for the experiment[9]:

H_{01} *An inspection team is as effective or more effective using CBR than it is using PBR.*

H_{02} *An inspection team using PBR finds defects at the same or higher cost per defect than a team using CBR for the defect detection phase of the inspection.*

H_{03} *An inspection team using PBR finds defects at the same or higher cost per defect as a team using CBR for the meeting phase of the inspection.*

H_{04} *An inspection team using PBR finds defects at the same or higher cost per defect than a team using CBR for all phases of the inspection.*

3 Research Method

This section describes the investigated reading techniques, the experimental design, and the techniques for analysing the collected data.

3.1 Implementation of Reading Techniques

3.1.1 Checklist-based Reading

The CBR approach attempts to increase inspection effectiveness and decrease the cost per defect by focusing the attention of inspectors on a defined set of questions. While there are existing checklists for code inspections (NASA, 1993), we have not been able to find any checklists to support defect detection in object-oriented design documents that we could reuse for the purpose of this experiment. Hence, we had to develop the checklist from scratch. We structured the checklist according to the schema that is presented in (Chernak, 1996). The schema consists of two components: “Where to look” and “How to detect”. The first component is a list of potential “problem spots” that may appear in the documents and the second component is a list of hints on how to identify a defect in the case of each problem spot. In addition, we defined the required quality properties for object-oriented design documents at the beginning

[8] One can argue that more time could be taken during the meeting for an inspector to understand the other perspective and understanding aspects of the design not demanded by his/her own perspective. This could result in a greater meeting time for PBR than for CBR. However, in such a case one would expect inspectors to share the extracted and documented information during their individual reading, making it easier for an inspector to comprehend the reasoning behind the other perspective’s defects. For CBR such extracted and documented information is not available.

[9] In the experiment, we investigate directional null hypotheses. A directional null hypothesis can be regarded as a statement that in the population there is a difference opposite to that predicted between the two populations (Aron and Aron, 1994). According to the logic of testing statistical hypotheses (Winer et al., 1991), we are interested in being able to reject these null hypotheses. The hypotheses are preceded by a zero to indicate that these are the null hypotheses being tested.

of the checklist so that the inspectors had a common understanding of those terms when we used them in the checklist. Finally, we added a check box so that an inspector could mark a specific question once he or she had performed the required checks. Figure 2 depicts an excerpt of the checklist (without the checkboxes). The complete checklist can be found in the Appendix.

There are three basic criteria for evaluating the quality of UML-design diagrams: Correctness, Completeness, and Consistency. Each one of these criteria can be operationalized as follows:

Correctness
A diagram or a set of diagrams is correct if it is judged to be equivalent to some reference standard that is assumed to be an infallible source of truth. In the case of design diagrams, the analysis diagrams can be considered as such.

Completeness
A diagram or a set of diagrams is complete if no required elements are missing. It is judged by determining if the entities in the (set of) diagrams describe the aspects of knowledge being diagramed in sufficient detail for the part of the system under development.

Consistency
A diagram or a set of diagrams is consistent if there are no contradictions among its elements. This may be judged by checking the consistency between the elements of a single diagram or between several diagrams/ diagram types

Item no.	Where to look	How to detect
1	**All Design Diagrams**	Is each name unique?
2		Are all names used in the diagrams consistent?
3		Are all names used in the diagrams correct?
4	**Collaboration Diagrams**	Are the parameters and parameter types correct when compared to the “reads” and “changes” clause in the operation schemata?

Figure 2: Example Questions of the Checklist.

3.1.2 Perspective-Based Reading

The basic idea behind the PBR approach is to inspect an artefact from the perspectives of its individual stakeholders. Since different stakeholders are interested in different quality factors or see the same quality factor quite differently (McCall, 1994), a software artefact needs to be inspected from each stakeholder’s viewpoint. The goal of inspectors applying the PBR technique is therefore to examine the documents of a software product from the perspectives of the product’s various stakeholders for the purpose of identifying defects.

For examining the documents from a particular perspective, the PBR technique provides guidance for an inspector in the form of a PBR scenario on how to read and examine the document. A scenario is an algorithmic guideline on how inspectors ought to proceed while reading the documentation of a software product, such as the design documents of a software system.

As depicted in Figure 3, a scenario consists of three major sections: introduction, instructions, and questions. This scenario structure is similar to the one described in the existing formulation of PBR for requirements documents (Basili et al., 1996).

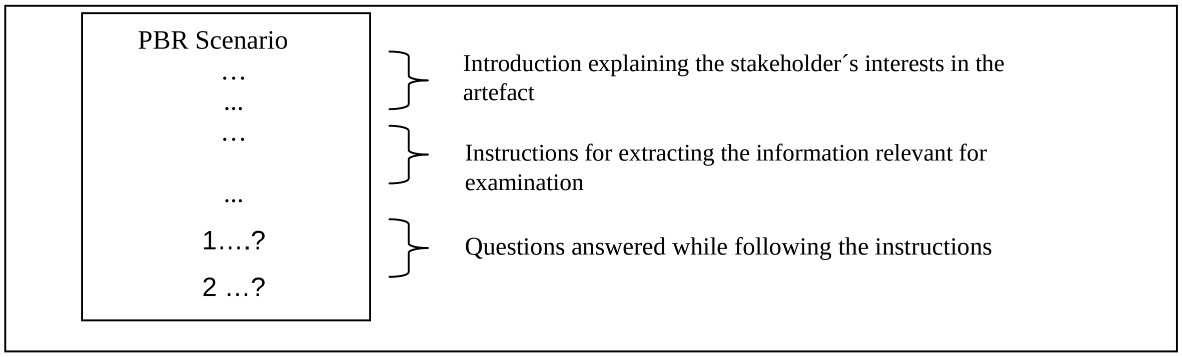
PBR Scenario
…
...
…
...
1….?
2 …?
Introduction explaining the stakeholder´s interests in the artefact
Instructions for extracting the information relevant for examination
Questions answered while following the instructions

Figure 3: Content and Structure of a PBR Scenario.

The introduction describes a stakeholder's interest in the artefact and may explain the quality requirements most relevant for this perspective.

The instructions describe what kind of documents an inspector is to use, how to read the documents, and how to extract the appropriate information from them. While identifying, reading, and extracting information, inspectors may already detect some defects. However, an inspector is to follow the instructions for three reasons. First, instructions help an inspector decompose large documents into smaller parts. This is crucial because people cannot easily understand large documents. Understanding involves the assignment of meaning to a particular document or parts of it and is a necessary prerequisite for detecting more subtle defects, which are often the expensive ones to remove if detected in later development phases. In cognitive science, this process of understanding is often characterised as the construction of a mental model that represents the objects and semantic relations in a document (van Dijk and Kintsch, 1984). Second, the instructions require an inspector to actively work with the documents. This ensures that an inspector is well prepared for the following inspection activities, such as the inspection meeting. Finally, the attention of an inspector is focused on the set of information that is relevant for one particular stakeholder. The particular focus avoids the swamping of inspectors with unnecessary details.

Once an inspector has achieved an understanding of the artefact, he or she can examine and judge whether the artefact as described fulfils the required quality factors. For making this judgement, a set of questions focus the attention of an inspector on specific aspects of the artefact, which can be competently answered because of the attained understanding.

3.1.3 Perspective-based Reading of Object-Oriented Design Documents

In the context of our study, design documents of a software system were inspected. We identified three perspectives for their inspection: A designer perspective, a tester perspective, and an implementer perspective. An inspector reading the design documents from the point of view of a designer is primarily interested in the correctness between the design models and the analysis documents. Hence, he or she has to extract relevant information from the design documents and compare it to the one described in the analysis documents. Crucial deviations in the functionality are considered potential candidates for defects. An inspector reading the design documents from the perspective of a tester identifies the different operations that the system is to perform in the design models and tries to set up test cases with which he or she can ensure the correct behaviour of each operation. Then, the inspector is supposed to mentally simulate each operation using the test cases as input values and to compare the resulting output to the description in the analysis documents. Any deviation points to a potential defect. Finally, an inspector reading the design documents from the perspective of an implementor makes sure that all required information is provided in the design models to implement the system. This involves completeness checks as well as more difficult checks on the feasibility of the design.

For each of the perspectives we developed a scenario according to the process outlined in (Laitenberger and Atkinson, 1999). The scenarios can be found in the Appendix. We hypothesize that actively working with the design document in this manner improves an inspector's understanding of the system from the particular perspective, which in turn helps him or her detect defects.

In the context of our experiment, each inspection participant executes one specific scenario. We expect the sum of the perspectives (i.e., the result of an inspection team) to provide high defect coverage. This is why we also performed inspection team meetings as part of the experiment. In this context, an inspection team in a PBR inspection consists of three inspectors, each of whom has read the documentation of a system from a different perspective.

3.2 Experimental Design

In this section we describe the rationale behind the study design that we employed, the subjects, the measurement approach, and the materials used.

3.2.1 Description of the Environment and the Subjects

The experiment was performed in the context of a course on object-oriented development. The main objective of this course was to teach participants the principles of object-oriented analysis, design, and development of information systems. The course consisted of several modules, each of which addressed a specific topic in object-oriented development, e.g., object-oriented analysis, object-oriented design, UML, software inspection, testing, etc. Each module included practical exercises in the form of case studies to familiarize the participants with the concepts being taught.

We performed the experiment in the module on software inspections. This module was scheduled towards the end of the course. This ensured that the participants already had theoretical and practical training in the analysis and design of information systems as well as in the UML. As a result we could assume that the subjects were familiar with the analysis and design phases as well as the notation used in the study materials.

The subjects in this experiment were 18 practitioners with various backgrounds. Before the course they primarily worked as programmers in industry and had various levels of experience in object-oriented programming. We captured the subjects' experience with a questionnaire before the study. We considered the subjects' experience in UML, design, programming, and inspections as the most relevant types of experience that would have an impact on defect detection capabilities. We found that most of the subjects had already written UML documents (median of 3 on a 5 point scale). Moreover, they had some experience in developing design documents and between 1 and 16 years of programming experience with a median of 2 years. None of them had yet participated in an inspection. Figure 4 depicts the distribution of experience in UML, design, and programming.

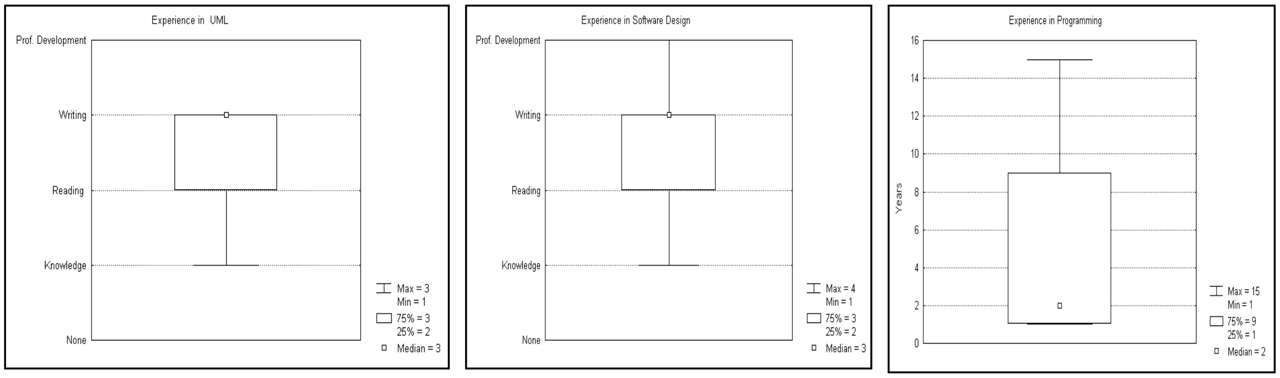

Figure 4: Histogram of Subjects' Experience in UML, Design, and Programming.

The term "Knowledgeable" means that their knowledge is based on the course content. The term "Prof. Development" refers to large-scale professional development in an industrial setting.

3.2.2 Experimental Materials

In order to use a set of diagrams to develop a system, it is necessary to follow a process. The development process defines the sequence of steps for building the various UML models. Moreover, a process relates the different models and diagrams to each other. This means that a development process must make it clear which documents (e.g., models, diagrams) contain relevant information about which software entities.

Of the leading object-oriented methods in widespread use, the one which best meets this criterion is the Fusion method (Coleman et al., 1994)[10]. Fusion is very precise about which specific models should be created as part of an object-oriented development project, and what information these models should contain. In contrast, other object-oriented methods typically give little prescriptive advice about what exactly to create during a project, and which activities to perform. As a consequence, when inspecting an entity, it is not easy to ensure that all the relevant information (i.e., models) describing properties of the entity have been found and checked. Fusion also has the advantage of using a mix of textual and graphical models, and therefore reinforces the idea that the documents used and identified for perspective-based reading can be of any kind. For these reasons, we use Fusion as the basis of the experimental materials.

Although Fusion uses diagram types that are quite similar to those of the UML (in fact, some of the UML models are derived from Fusion), these do conform precisely to the UML standard. Therefore, in the experiment we used a hybrid version of the Fusion method, known as FuML (Atkinson, 1998), which specifically adapts Fusion to exploit the UML. The mapping of Fusion to UML models adopted in FuML is given in Figure 5. An important point to notice in this mapping is that there is no UML counterpart for Fusion operation schemata, which serve to define the functional properties of individual system operations, since these are textual models. The UML standard deliberately avoided the inclusion of textual models. Therefore, the model inspected as part of the experiment included Fusion operation schemata in addition to UML diagram types.

The FuML approach calls for a system to be described at two main levels of abstraction: the analysis level and the design level. The analysis level includes a class diagram (known as the system class diagram), a use case diagram, a set of sequence diagrams (one for each use case), a set of operation schemata (one for each externally visible operation of the system) and a short textual description. The design level, on the other hand, includes a more detailed class diagram (known as the design class diagram) and a set of collaboration diagrams (one for each operation of the system). All but the operation schemata and the textual descriptions are presented as UML diagrams.

[10] The most popular object-oriented development process is probably the Rational Unified Process (RUP) (Jacobson et al., 1998). However, the Fusion process is more precise than RUP when it comes to defining when to create the various models and which set of models to create as part of the development effort.

Fusion		UML
Analysis		
Object-Model	⇒	Class diagram(s)
Operation Model	⇒	No UML counterpart
Life-cycle Model	⇒	State diagram
Scenarios	⇒	Sequence diagrams
Design		
Object Interaction Graphs	⇒	Collaboration diagram(s)
Visibility Graphs	⇒	Associations in class diagram(s)
Class Descriptions	⇒	No UML counterpart
Inheritance Graphs	⇒	Inheritance in class diagram(s)

Figure 5: Fusion Model Substitution

For the experiment we developed these models for two different systems[11]. The first one is a web-based quiz system that allows its users to perform a quiz using the web. The second one is a point of sales system. For each of these systems, we provided the subjects with the system's analysis and design documentation. The size of the web-based quiz system documentation was 21 pages. The quiz system included 6 collaboration diagrams and 4 design class diagrams. The size of the point of sales system was 18 pages. The point of sales system included 6 collaboration diagrams and 3 design class diagrams.

We introduced a set of defects into each of the design documents prior to the experiment. We inserted 21 defects in the quiz system and 19 defects in the point of sales system. The defects primarily related to correctness, consistency, and completeness of the design models. Some of the defects were made while developing the documents. However, we introduced some more defects in the design documents to have a larger set of defects. We perceived these defects to be realistic for UML designs based on our collective experience.

To ensure the feasibility of the required reading activities and the possibility for subjects to scrutinize each of the documents for defects in a 2-4 hour time frame, we performed a trial run of the document inspection. As part of the trial run, we used the different reading scenarios as well as the checklist ourselves to scrutinize the documents for defects. After the trial run, we improved the scenarios, the checklist, and the documents based on our own experience in using them. This included the removal of spelling mistakes we found in the examples.

[11] The documents are available from the first author.

3.2.3 Constraints for the Experimental Design

In our particular context, three constraints were important influences on the overall experimental design:

Inability to Withhold Treatment. Since our study was performed in the context of a training course, each subject had to learn and apply both reading techniques. It has been noted that withholding treatment may contribute to demotivation and subsequent confounding of the treatment effects. This immediately suggests a repeated-measures design. Even though it is dictated by the study constraints, a repeated-measures design has certain additional advantages over a between-subjects design. Repeated-measures designs have higher statistical power. This is because there will almost always be a positive correlation between the treatments (Keren, 1993). Previous empirical studies of different aspects of developer performance have found that individual performance differences can vary from 4 to 1 to 25 to 1 across experienced developers with equivalent backgrounds (Brooks, 1980). The high subject variability can easily mask each treatment effect that is imposed on the subjects in an experiment. This has caused some methodology writers to strongly recommend repeated-measures designs, since subjects effectively serve as their own controls (Brooks, 1980). Repeated-measures designs enable a direct and unconfounded comparison between the different treatments (Keren, 1993). Finally, repeated-measures designs have an economical advantage in that less subjects are required compared to a between-subjects design to attain the same statistical power levels (Lipsay, 1990).

No Control Group. The validity of the concept of a “no-treatment” control group in software engineering research has been questioned (Kitchenham et al., 1994). This is because it is not clear what a “no-treatment” group is actually doing. A suggested alleviation of this in an industrial setting is that the “no-treatment” group performs the reading technique that they usually employ in their practice. Indeed, this was the approach followed in the study with professional engineers at NASA/GSFC (Basili et al., 1996). In our current study, we only compared the two reading techniques CBR and PBR. This implies that it is impossible to determine whether either one or both of the reading techniques are better than the current practice of the subjects. From a general scientific perspective this issue is not of concern, since the questions being answered are related to the relative performance of the reading techniques and we wish to draw general conclusions beyond the specific daily practices of the subjects.

Counterbalancing. A danger with repeated-measures designs is the existence of carry-over effects (Greenwald, 1976). Carry-over effects can occur in two ways: Practice effects and sequence effects. With the former, treatment effects are confounded with practice. In our context, practice can occur because the subjects had not used a systematic reading technique in the past. Therefore, it is plausible that with the second treatment they will exhibit better performance than with the first, because they had practice with a reading technique in the first treatment, irrespective of what the reading technique was.

When confronted with a sequence effect, the effect of the first treatment persists and thus contaminates the measurements on the second treatment. Applying a prescriptive reading technique before a “usual” reading technique can lead to carry-over effects. This is because the subjects would not be able to completely stop using the prescriptive technique. Since in our context one can argue that both techniques are prescriptive, this caution would not be applicable.

To err on the conservative side, and also to address the potential dangers of practice effects, we used a counterbalanced repeated-measures design. With counterbalancing both combinations by which treatments can be ordered are used. In our experiment we had two groups constructed randomly. Each of the 2 groups was randomly assigned to one of the two sequences. The effect of counterbalancing is to spread the unwanted variance arising from the treatment by practice or sequence interaction amongst the different treatments.

Furthermore, an explicit test for carry-over effects is conducted during our analysis to check if any such effects influence our results despite the counterbalancing.

3.2.4 The Counterbalanced Repeated-Measures Design

Our final experimental design is depicted in Table 2. We use a notational system in which X stands for a treatment and O stands for an observation; subscripts 1 and 2 refer to the sequential order of implementing treatments. For the experiment, the subjects were randomly assigned to two groups[12]. The first group performed a reading exercise using PBR first, and then measures were collected. Subsequently, they performed a reading exercise using CBR, and again measures were collected. The second group followed the counterbalanced sequence. This is a classic design described more fully in (Winer et al., 1991).

Experiment				
Group 1	X_{PBR}	O_1	X_{CBR}	O_2
Group 2	X_{CBR}	O_1	X_{PBR}	O_2

Table 2: The design of the experiment.

Below we discuss a number of issues related to the experimental design and its execution:

Different Design Documents. Given that each group performs two reading exercises using different reading techniques, it is impossible for a group to read the same design documents twice, otherwise they would remember the defects that they found during the first reading, hence invalidating the results of the second reading. Therefore, a group reads a different design document each time.

Equivalence of Design Documents. Since two documents are used in the experiment, we can control the document effect using one of two approaches. First, it can be explicitly controlled in the experimental design by including "document" as another factor. The small number of subjects precluded this option.[13] Second, we ensure that the documents are equivalent. This is the option we followed.

We already pointed out that the documents were similar with respect to their size. However, the documents may be dissimilar with respect to their cognitive complexity. If this were the case, then it would introduce a confounding effect. To dilute this possibility, we looked at the structural properties (e.g., coupling) between the two design documents. Structural class properties have been hypothesized in the literature to affect cognitive complexity, which in turn impacts the fault proneness of the system (El Emam et al., 1999). Hence, if the structural properties are similar, the systems would have the same cognitive complexity. The question is which structural properties to consider. In a recent report El Emam et al. investigated the influence of 24 object-oriented metrics (covering coupling, cohesion, inheritance, and complexity) on fault-proneness (El Emam et al., 1999). All 24 metrics measure structural properties of the system. The study was performed with data from a large telecommunications system. They found that out of the 24 metrics only four are actually related to faults after controlling for size, and only two of these are useful for the construction of prediction models. The two selected metrics measure coupling. Similar results were found in a subsequent study on a Java application (El Emam and Melo, 1999). Hence, we collected the metric "Coupling between Object Classes" (CBO) as defined in (Chidamer and Kemerer, 1995). There, a class is coupled with another if methods of one class use methods or attributes of the other, or vice versa. This includes inheritance-based coupling. We calculated the mean CBO value for both systems. Both systems exhibit the same CBO value of 1.2. This means that the design documents have about the same cognitive complexity and the probability is rather low that there is a document effect.

To further verify the equivalence of the two design documents, we also did collect subjective information on the understandability of the two design documents from the subjects after the experiment. Figure 6 depicts the subjects' perception of the difficulty in understanding the design documents of the two systems.

[12] For the random assignment, the subjects drew numbers from an envelope.

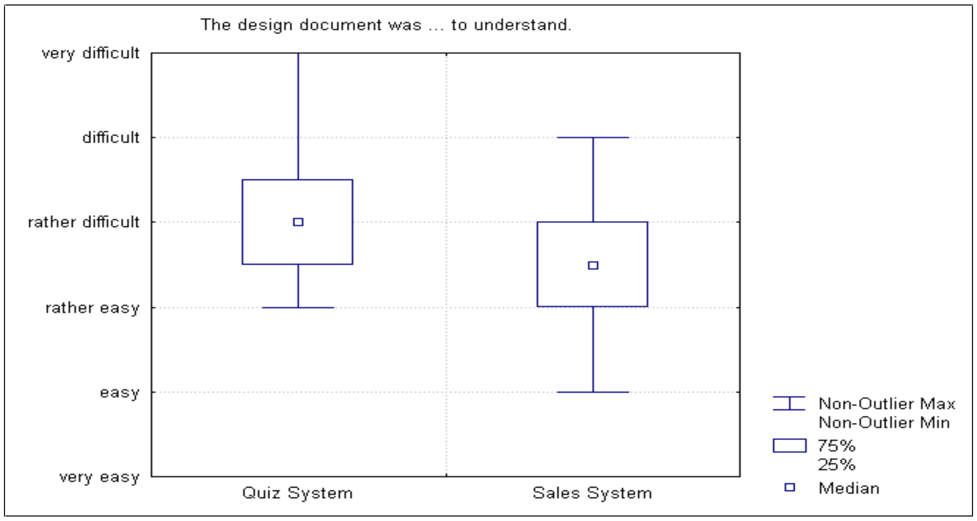

Figure 6: Difficulty in Understanding.

Although it seems that the quiz system is slightly more difficult to understand than the sales system, the difference is not statistically significant according to a Wilcoxon matched pairs test (p=0.17)[14]. Therefore, there is considerable evidence that the two design documents are equivalent in size and complexity, and there is no significant difference in their understandibility by the subjects.

Feedback Between Treatments. The subjects did not receive feedback about how well they were performing after the first treatment. This alleviates some of the problems that may be introduced due to learning (Basili et al., 1998). If subjects do not receive feedback, then they are less likely to continue applying the practices they learned during the first treatment.

Power Analysis. Ideally, researchers should perform a power analysis before conducting a study to ensure that their experimental design will find a statistically significant effect if one exists. However, in our case, such an a priori power analysis was difficult because the effect size is unknown. As mentioned earlier, there have been no previous studies that compared PBR to CBR for object-oriented design documents, and therefore it was impossible to use prior effect sizes as a basis for a power analysis. We therefore performed an a posteriori power analysis for all hypotheses using the obtained effect size. Table 3 presents the posteriori power levels.

	H_{01}	H_{02}	H_{03}	H_{04}
Experimental Power	0.83	>0.9	>0.9	>0.9

Table 3: A posteriori power analysis results.

Table 3 demonstrates that the power levels of our experiment were sufficiently high when considering recommendations in the literature (power levels of at least 0.8) as a kind of threshold value. This means that our experiment has a high probability of rejecting the null hypotheses if they were false (Cohen, 1988).

Process Conformance. It is plausible that subjects do not perform the CBR and PBR reading techniques but revert to an ad-hoc reading approach. We did not consider this a serious issue in our experiment

[13] Recall that the unit of analysis in our experiment is the team.

[14] Assuming a difference in understandability between the design documents, we expect that the teams detect more defects in the design document that is easier two understand, i.e., the Point of Sales System. However, we found that the teams had a slightly higher defect detection effectiveness when inspecting the Web-based Quiz System.

given that the subjects had little exposure to object-oriented analysis and design documents, i.e., without any defect detection aid they neither know what to look for nor how to perform the required checks. With PBR it is possible to check process conformance explicitly by examining the intermediate documents that are turned in. We did this, and determined that the subjects did perform PBR as defined. It is one of the disadvantages of CBR that such conformance checks cannot be performed easily. This is why, in this experiment, we introduced check-boxes in the checklists to investigate whether the subjects had performed the required checks. In most cases, the subjects marked all the questions that were included in the checklist. However, for CBR we cannot be sure that the subjects really performed the required checks. So, we can only assume that process conformance is also high when CBR is used.

3.2.5 Measurement

3.2.5.1 Dependent Variables

In this experiment we investigated four dependent variables: Team defect detection effectiveness and the cost per defect, with three different definitions. Team defect detection effectiveness refers to the number of defects reported by a two- or three-person inspection team (without defects found in the meeting). As the different design documents included a different number of defects, we had to normalize the detected number of defects. We did this by dividing the number of defects detected by the total number of defects that is known. Hence, the dependent variable “team defect detection effectiveness” can be defined in the following manner:

$$\text{Team defect detection effectiveness} = \frac{\text{Defects found by a three} - \text{person team}}{\text{Total number of defects in the design document}} \quad \textbf{Eqn. 1}$$

The cost per defect for teams is defined in three different ways depending on which phases of the inspection process are taken into account. The first definition relates the defect detection cost to the number of defects found by the inspection team. The second one relates the meeting cost to the number of defects found by the inspection team. And the third relates the sum of the defect detection cost and the meeting cost to the number of defects found by the inspection team. The dependent variable “cost per defect for teams” can be defined in the following manner:

$$\text{Cost per defect for the defect detection phase} = \frac{\text{Defect detection effort of three subjects}}{\text{Defects found by a three - person team (without meeting gains)}} \quad \textbf{Eqn. 2}$$

$$\text{Cost per defect for the meetinge phase} = \frac{\text{Meeting effort of three subjects}}{\text{Defects found by a three - person team (without meeting gains)}} \quad \textbf{Eqn. 3}$$

$$\text{Cost per defect for the overall inspection} = \frac{\text{Detection effort} + \text{Meeting effort}}{\text{Defects found by a three - person team (without meeting gains)}} \quad \textbf{Eqn. 4}$$

Below we address some issues related to the counting of the number of defects found:

Defect Reporting. The subjects sometimes reported more defects on their defect report forms than were seeded in the design documents. When a true defect was reported that was not on the list of seeded defects, we added this defect to the list of known defects and re-analyzed the defect report forms of all the remaining subjects.

Meeting Gains and Losses. For the evaluation of our hypotheses, data was collected after applying each reading technique and after the team meetings. The data collected after the teams may be distorted by meeting gains and losses that are independent of the reading technique used. We found very little meeting gains and meeting losses. We excluded meeting gains from the team results. The meeting losses were sufficiently minor that this issue was ignored in the analysis.

3.2.5.2 Independent Variables

We controlled two independent variables in the experiment:

- Reading technique (CBR versus PBR)
- Order of reading (CBR → PBR versus PBR → CBR).

3.2.6 Execution

The experiment was performed in August 1999. It consisted of four sessions and each session lasted 0.5 days. The experiment was conducted in the following manner (see Table 4). We started with an intensive exercise introducing the principles of software inspection and the Fusion development method. This included a brief explanation of the various UML models the subjects were to inspect. We then split the group in a random fashion. Depending on the reading order, we then explained in detail either CBR or PBR. This explanation covered the theory behind the different reading approaches as well as how to apply them in the context of an object-oriented design document. Then, the subjects used the explained reading approach for individual defect detection in the assigned design document. Regarding the PBR technique, each subject either used the designer, the implementor, or the tester scenario but not all three of them. The assignment of perspectives to inspectors was also performed in a random fashion.

While inspecting a design document, the subjects were asked to log all detected on a defect report form. After the reading exercise we asked the subjects to fill out a debriefing questionnaire. The same procedure was used on the second day for the reading approach not applied on the first day. After the reading exercises, we described to the subjects how to perform inspection meetings. To provide the participants more insight into an inspection meeting, we randomly assigned subjects to three-person inspection teams and let them perform inspection meetings (one for each design document). We ensured that within each of these teams one participant read a design document from the perspective of a designer, one read a design document from the perspective of a tester, and one read the design documents from the perspective of an implementor. Furthermore, all team participants had applied the reading techniques in the same order, and they had the meetings in the same order as the design documents were scrutinized for defects. The inspection team was asked to log all defects upon which all agreed. [15]

[15] An alternative execution of the experiment would be to have the training, reading, and meeting for each reading technique followed by the sequence for the other reading technique. This would have required five sessions, however, whereas our approach required only four sessions. Recall that the purpose of the meeting is to consolidate defects rather than to detect defects, therefore whether they meet in between defect detection session or after all defect detection sessions is not expected to play an important role. Furthermore, introducing a meeting in between the subjects' application of reading techniques would provide them with feedback about their performance. As noted earlier, we wished to avoid providing feedback during the experiment (only after) to avoid process conformance problems.

	Day 1		**Day 2**	
	Morning	Afternoon	Morning	Afternoon
Group 1	Introduction of Inspection Principles	PBR Explanation Defect Detection with PBR	CBR Explanation Defect Detection with CBR	Team Meetings
Group 2	Introduction of Inspection Principles	CBR Explanation Defect Detection with CBR	PBR Explanation Defect Detection with PBR	Team Meetings

Table 4: Execution of the Experiment.

3.3 Analysis Methods

For the analysis of the collected experimental data, we used the match-paired t-test to investigate our hypotheses (Aron and Aron, 1994). [16] Since we investigated directional hypotheses, we used a one-tailed test. For hypothesis testing, we used a standard α-level of 0.05.

One potential difficulty associated with this approach is the fact that the t-test makes specific assumptions about the population under study and that our sample size is small. The t-test assumes, for example, that the data is distributed normally. One solution would be to use exact permutation tests (Metha and Patel, 1999). These make no distributional assumptions, and provide exact p-values even for small samples.[17] Furthermore, permutation tests do not require assumptions about random sampling from a population (Edgington, 1980). We calculated the exact p-values for a permutation test on the (robust) median using a permutation test in addition to the t-test. In all cases the conclusions were the same as the t-test.

Our experimental design permits the possibility of carry-over effects. Grizzle (Grizzle, 1965) points out that when there are carry-over effects from treatment 1 to treatment 2, it is impossible to estimate or test the significance of any change in performance from Period 1 and Period 2 over and beyond carry-over effects. In that situation, the only legitimate follow-up test is an assessment of the differences between the effects of the two treatments using Period 1 data only. Hence, it is recommended that investigators first test for carry-over effects[18]. Only if this test is not significant, a further analysis of the total data set is appropriate. In that case, all the obtained data may properly be used in separate tests of significance for treatments.

4 Analysis Results and Interpretation

This section presents and discusses the results of the experiment in terms of the defect detection effectiveness, the cost per defect for the defect detection phase, the cost per defect for the meeting phase, and the cost per defect for the overall inspection. It also provides sample size requirements for future studies.

[16] During the course of the study, two subjects had performance that was very different from the rest of the experimental subjects. This was attributed to their lack of experience. We performed the analysis with the subjects included and with them excluded, and the conclusions are the same. We therefore present only the results with the values for these subjects excluded. Because we performed a within-team comparison, these subjects do not a prori impact the results of one reading technique more than the other.

[17] The data are permuted repeatedly between treatments, and for each permutation of the data a test statistic is computed under the null hypothesis to determine the proportion of the data permutations that provide as large a test statistic value as that observed in the experiment. If that proportion is as small as significance level α or smaller, the results are significant at the α-level. More information on these tests can be found in (Good, 1994).

[18] We used a standard α-level of 0.05 for testing whether there is a carry-over effect.

4.1 Defect Detection Effectiveness

Figure 7 depicts the defect detection effectiveness of the inspection teams when applying the two different reading techniques.

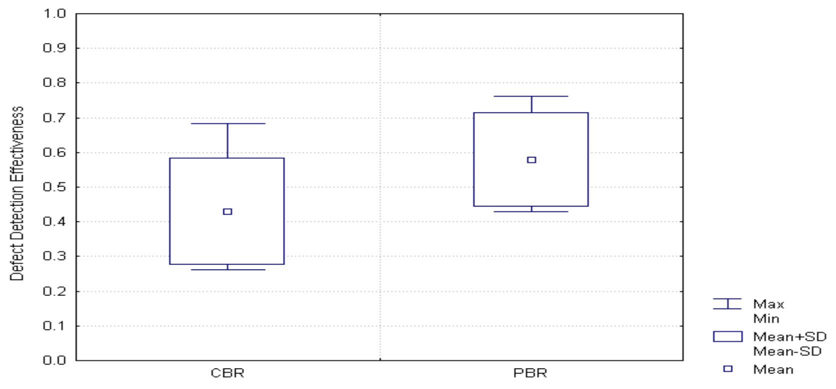

Figure 7: Defect Detection Effectiveness.
(Mean values: CBR: 0.43; PBR: 0.58)

An inspection team usually found more defects using the PBR technique than using the CBR technique. Using a matched-pair t-test the difference is statistically significant (t=3.17, p=0.025). The permutation test revealed an exact p-value of 0.0313. We also failed to detect a carry-over effect for the different sequences. This means that the sequence of the various reading techniques did not influence the results of this study.

We also calculated the average percentage improvement of PBR over CBR. This can be calculated using the following formula.

$$\text{Relative Improvement for the Defect Detection Effectiveness} = \frac{\text{Defect Detection Effectiveness (PBR)} - \text{Defect Detection Effectiveness (CBR)}}{\text{Defect Detection Effectiveness (CBR)}}$$

The average relative improvement of all inspection teams is 0.41. This means that PBR exhibits a 41 percent effectiveness improvement over CBR.

Based on these findings, we can reject hypothesis H_{01} and conclude that an inspection team is more effective using the PBR technique than using CBR.

4.2 Cost per Defect for the Defect Detection Phase

Figure 8 depicts the cost per defect ratio of the defect detection phase when using the two different reading techniques.

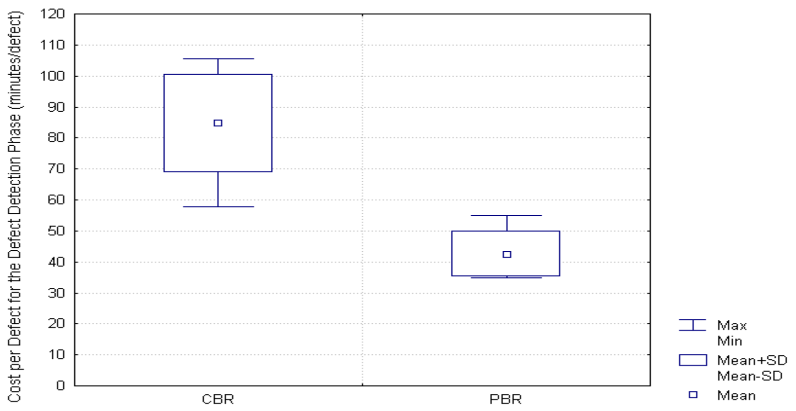

Figure 8: Cost per Defect for the Defect Detection Phase.
(Mean values: CBR: 85 minutes per defect; PBR: 43 minutes per defect)

Teams had a lower cost per defect ratio using the PBR technique than using the CBR technique when considering the effort for the defect detection phase. Using a matched-pair t-test the difference is statistically significant (t=-6.38, p=0.001). These findings were also confirmed with the permutation test. The permutation test revealed an exact p-value of 0.0156. We failed to detect carry-over effects. The study results therefore suggest that the PBR technique improves the cost-effectiveness of inspection teams when considering the defect detection effort.

Based on these findings, we can reject hypothesis H_{02} stated in Section 3.

4.3 Cost per Defect for the Meeting Phase

Figure 9 depicts the cost per defect ratio of the meeting phase when using the two different reading techniques.

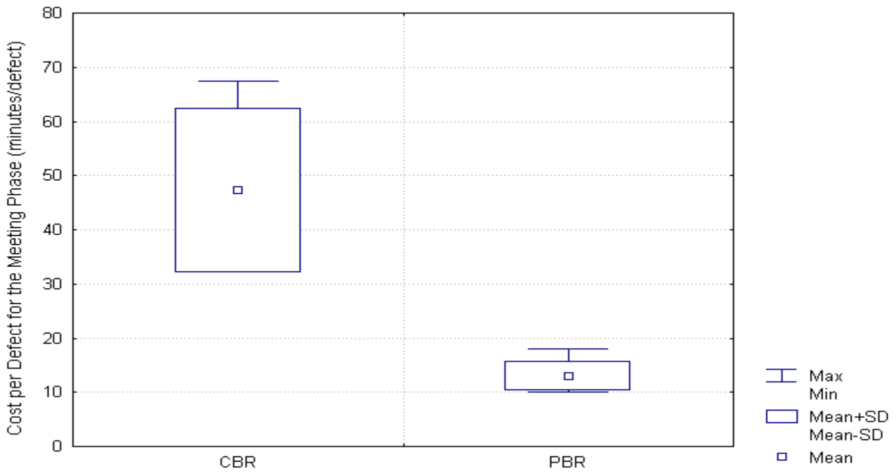

Figure 9: Cost per Defect for the Meeting Phase
(Mean values: CBR: 47 minutes per defect; PBR: 13 minutes per defect)

The team with the PBR technique had a lower cost per defect ratio than using the CBR technique when considering the meeting effort. Using a matched-pair t-test the difference is statistically significant (t=-5.98, p=0.002). The permutation test revealed an exact p-value of 0.0156. We failed to detect carry-over effects. The study results therefore suggest that the PBR technique improves the cost-effectiveness of inspection teams when considering the meeting effort.

Based on these findings, we can reject hypothesis H_{03} stated in Section 3.

4.4 Cost per Defect for the Overall Inspection

Figure 10 depicts the cost per defect ratio of the overall inspection when using the two different reading techniques.

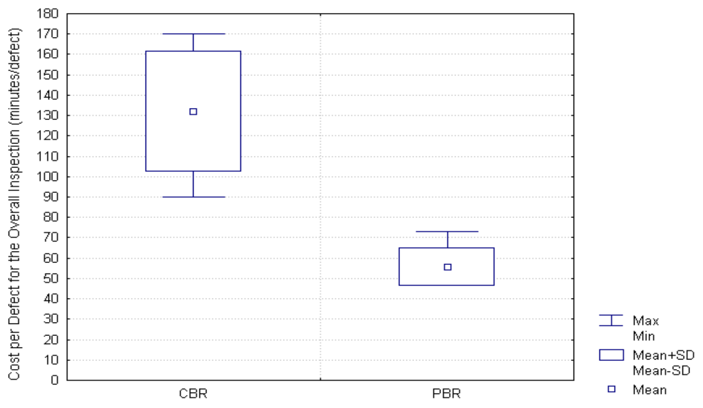

Figure 10: Cost per Defect for the Overall Inspection
(CBR: 132 minutes per defect; PBR: 56 minutes per defect).

The team with the PBR technique had a lower cost per defect ratio using the CBR technique than using the PBR technique when considering the effort for the overall inspection. Using a matched-pair t-test the difference is statistically significant ($t=-6.53$, $p=0.001$). These findings were also confirmed with the permutation test ($p=0.0156$). We failed to detect carry-over effects. The study results therefore suggest that the PBR technique improves the cost-effectiveness of inspection teams.

We also calculated the average percentage improvement of PBR over CBR. This can be calculated using the following formula.

$$\text{Relative Improvement for the Cost per Defect} = \frac{\text{Cost per Defect (PBR) - Cost per Defect (CBR)}}{\text{Cost per Defect (CBR)}}$$

The average relative improvement of all inspection teams is 0.58. This means that PBR exhibits a 58 percent cost per defect improvement over CBR.

Based on these findings, we can reject hypothesis H_{04} stated in Section 3.

4.5 Discussion

The findings reveal that, on average, the inspection teams found more defects using the PBR technique than using the CBR technique (average increase of 41% in effectiveness). We did not impose any time constraint on the subjects, i.e., the subjects had as much time available as needed to complete their scrutiny. This allowed us to also investigate the cost per defect ratio. We found that the subjects consumed more effort for the CBR approach than for the PBR approach (average reduction of 58% in cost per defect). On average, the teams applying the PBR technique detected 58 percent of the defects and required 56 minutes to detect a single defect. These results are consistent with previous findings comparing PBR with CBR for the inspection of C code (Laitenberger et al., 2000).

We also asked the subjects about the difficulty of using the two reading techniques. Figure 11 depicts the results of this question.

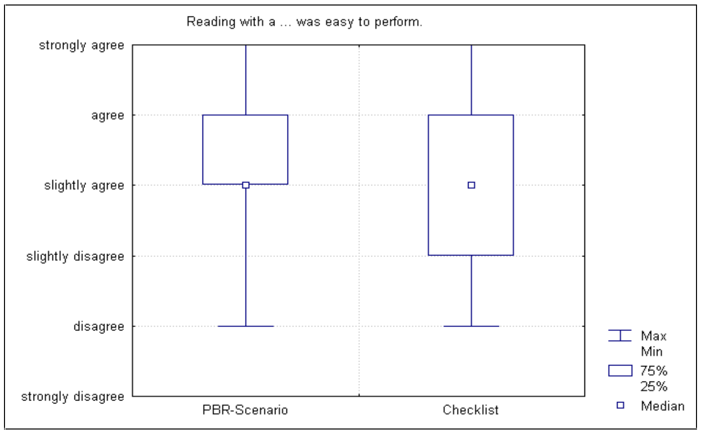

Figure 11: Ease of Use.

As depicted in Figure 11 the subjects perceive both reading techniques quite similar with respect to their ease of use. This finding together with the better effectiveness and cost per defect values for PBR convinced us that it may be more valuable for companies to use the PBR technique rather than a checklist for defect detection in their software inspections.

4.6 Sample Size Requirements for Future Studies

The planning of future studies that compare CBR to PBR can benefit from the estimates of effect size that we obtained. For repeated measures designs, we used the obtained mean effect sizes[19] and the average correlation coefficients to estimate the minimal number of teams that would be necessary to attain a statistical power of 80 percent for one-tailed tests at an alpha level of 0.05 using the paired t-test. These are summarized in Table 5.[20]

	Team Defect Detection Effectiveness	Cost per Defect (Defect Detection)	Cost per Defect (Meeting)	Cost per Defect (Overall Inspection)
Mean Effect Size	1.71	3.03	3.47	3.21
Estimated Sample Size	5	3	3	3

Table 5: Estimation of sample size (number of teams).

The above table reveals that in fact, small sample experiments would have sufficient power to detect such a large effect. This is encouraging for future research in that the cost of PBR comparative experiments could be kept low.

[19] We used Hedges g as our measure of effect size (Hedges and Olkin, 1995). However, we had to consider that our experiment uses a "within-subject" design meaning that the two samples being compared are not independent of each other. Hence, we considered the strength of the correlation $r_{PBR/CBR}$ between the paired values in the calculation of the effect size (Lipsey, 1990).

[20] These sample size estimates are for three-person inspection teams. It is plausible that if there are more than three inspectors the effect size will be larger. Therefore, if one utilizes the above sample sizes in planning a study, they are certain to attain 80% power for a study with more than three inspectors.

5 Threats to Validity

This section discusses the threats to internal and external validity in addition to the experimental constraints that we have discussed previously. Threats to internal validity are unknown or uncontrollable factors that impact the results of the experiment. Threats to external validity influence the generalizability of the findings (Winer et al., 1991).

5.1 Threats to Internal Validity

There are two potential threats to the internal validity of our experiment: learning and equivalence of design documents. We discuss how these were addressed below.

Learning effects can appear in many different ways in an experiment. The learning effect is a particular danger for within-subject designs. First, there may be a learning effect because of the UML notation. This means that subjects increase their performance in the experiment because they learn the semantics of the notation. However, the subjects participated in many training sessions beforehand. Hence, we can assume that they were quite familiar with the notation used in the document. Furthermore, the counterbalancing would distribute such an effect across both reading techniques, hence reducing its impact. The second learning effect may be due to the reading technique. The subjects increase their performance in the second session because they remember the technique of the first one. Again, to counteract this threat to validity, we used a counterbalanced design. Moreover, we investigated whether there are carry-over effects from one session to the next, but did not detect any.

In the design of this experiment we paid special attention to ensuring that the two design documents used by each subject were equivalent. If this was not the case, then there is potentially a confounding effect from the document (either directly or through its interaction with the treatment).[21] We presented evidence showing that the two documents were equivalent in size, and in terms of object-oriented structural properties that have been demonstrated to have an impact on fault-proneness through increased cognitive complexity. Furthermore, the subjects perceived no difference in understandability between the two documents. This gives us confidence that the potential for a confounding effect is minimal.

5.2 Threats to External Validity

We consider three potential threats to the external validity of our results: the subjects, the design documents, and the Fusion development process.

Our subjects may not be representative of the pool of software developers that professionally uses the UML for the analysis and design of object-oriented systems. However, they were all professional developers rather than students.

The design documents are not necessarily representative of the ones used in industry. The limitation primarily derives from the size of the created design documents. Systems in industry are usually much larger in size than the ones we used in this experiment. However, we regard the amount of material that our subjects were required to inspect in a single inspection as appropriate.

The design documents were developed according to the Fusion development process. Although this process is used at Hewlett-Packard (Coleman et al., 1994), other companies may follow another development process, e.g., the Rational Unified Process (Jacobson et al., 1998). Since all the Fusion models apart from operation schemata can be found in other UML-based development processes as well, we believe that this represents a rather limited threat to validity.

[21] For example, in previous experiments where two documents were inspected in a repeated-measures design, it was noted that there was a potential for a document by treatment confounding effect (Basili et al., 1996, Basili et al., 1998; Laitenberger et al., 2000). By ensuring document equivalence, this threat can be diminished.

6 Conclusion

Software inspection is regarded as one of the most effective methods for software quality improvement. To fully fulfil this role, however, a software inspection must involve a through and detailed examination of the inspected document. This requires reading techniques that tell inspection participants what to look for and how to scrutinize software documents in a systematic manner. Unfortunately, few reading techniques are available for defect detection in documents created according to object-oriented principles.

In this paper, we have elaborated upon the perspective-based reading approach for object-oriented design documents and compared its effectiveness and its cost per defect ratio to checklist-based reading. The comparison was performed through a controlled experiment with practitioners participating in a course on object-oriented development. During the experiment the subjects used the CBR approach as well as the PBR approach for defect detection in design documents. The design documents were specified in the UML.

Our results indicate that the effectiveness of teams using PBR is greater than of those using CBR. Furthermore, we found that the cost per defect ratio using PBR is smaller than with CBR during the defect detection phase of inspections. Applying a PBR scenario helps improve a subject's perceived understanding of the specified system. The increased understanding leads to a lower cost per defect for PBR compared with CBR during the meeting phase of an inspection. Overall, we found that the cost per defect for the whole inspection is lower with PBR than with CBR. Therefore, PBR has effectiveness and cost advantages when compared to CBR.

We have provided the checklist as well as the scenarios that we have used during the experiment in the Appendix to this paper. Organizations wishing to introduce perspective-based inspections can take these perspectives and scenarios as a starting point for tailoring the PBR technique to their specific environment. The specificity and the detail of the guidelines can then be further refined, and other perspectives defined, based on feedback from inspectors using these scenarios.

We are aware that experiments by their nature have difficulties in ruling out internal as well as external threats to validity. Some threats have their origin in the fact that experiments in software engineering usually involve a low number of subjects. We have been careful to describe and address all of these threats in detail so that other researchers can benefit from the lessons we have learned. They can, as a consequence, try to avoid these threats while replicating this experiment or setting up other empirical studies in this area.

To our best knowledge, this study represents the first controlled experiment that deals with UML documents. Hence, further empirical research is warranted to establish greater external validity for these results. We therefore encourage the external replication of this study in different environments by different researchers to build a significant body of knowledge (Brooks et al., 1996; Basili et al., 1998).

Acknowledgements

We want to extend our very warm thanks to the participants of this experiment. We also wish to thank the anonymous reviewers for their valuable comments that have contributed to improving the content and presentation of this paper.

7 References

Aron, A., and Aron, E., 1994. Statistics for Psychology. Prentice Hall, 1st edition.

Atkinson, C., 1998. Adapting the Fusion Process to Support the Unified Modeling Language. Object Magazine, p.32-39.

Basili, V., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S., and Zelkowitz M., 1996. The Empirical Investigation of Perspective-based Reading. Empirical Software Engineering, 2(1):133–164.

Basili, V., Shull, F., and Lanubile, F., 1998. Using Experiments to Build a Body of Knowledge. Technical Report, University of Maryland, CS-TR-3983.

Basili, V., 1997. Evolving and Packaging Reading Technologies. Journal of Systems and Software, 38(1):3-12.

Booch, G., 1993. Object-Oriented Analysis and Design with Applications. Benjamin Cummings.

Booch, G., Rumbaugh, J., Jacobson, I., 1999. The Unified Modeling Language User Guide. Addison-Wesley.

Briand, L., El Emam, K., Fussbroich, T., and Laitenberger, O., 1998. Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects. Proceedings of the 20th International Conference on Software Engineering, pages 340–349. IEEE Computer Society Press.

Brooks, A., Daly, J., Miller, J., Roper, M., and Wood, M., 1996. Replication of experimental results in software engineering. International Software Engineering Research Network (ISERN) Technical Report ISERN-96-10, University of Strathclyde.

Brooks, R., 1980. Studying Programmer Behavior Experimentally: The Problems of Proper Methodology. Communications of the ACM, 23(4):207–213.

Cheng, B., and Jeffery, R., 1996. Comparing Inspection Strategies for Software Requirements Specifications. Proceedings of the 1996 Australian Software Engineering Conference, pages 203–211.

Chernak, Y., 1996. A Statistical Approach to the Inspection Checklist Formal Synthesis and Improvement. IEEE Transactions on Software Engineering, 22(12):866–874.

Chidamber, S., and Kemerer, C., 1995. A Metrics Suite for Object-Oriented Design, IEEE Transactions on Software Engineering, 21(3):265.

Cohen, J., 1988. Statistical Power Analysis for the Behavioural Sciences. Lawrence Erlbaum Associate Publishers, 2nd edition.

Coleman, D., Arnold, P., Godoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P., 1994. Object-Oriented Development: The Fusion Method. Englewood Cliffs, NJ:Prentice Hall.

Dyer, M., 1992. The Cleanroom Approach to Quality Software Development. John Wiley and Sons, Inc..

Edgington, E.S., 1980. Randomization Tests. Dekker.

El Emam, K., Benlarbi, S., Goel, G., Rai, S., 1999. A Validation of Object-Oriented Metrics. Technical Report of the National Research Council of Canada, NRC/ERB-1063. NRC 43607. Submitted for publication.

El Emam, K., and Melo, W., 1999.The Prediction of Faulty Classes Using Object-Oriented Design Metrics. Technical Report of the National Research Council of Canada, NRC/ERB-1064. NRC 43609. Submitted for publication.

Fagan, M., 1976. Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, 15(3):182–211.

Fagan, M., 1986. Advances in Software Inspections. IEEE Transactions on Software Engineering, 12(7):744–751.

Fowler, P., 1986. In-process Inspections of Workproducts at AT&T. AT&T Technical Journal, 65(2):102–112.

Fusaro, P., and Lanubile, F., 1997. A Replicated Experiment to Assess Requirements Inspection Techniques. Empirical Software Engineering, 2(1):39–57.

Gilb, T., and Graham, D., 1993. Software Inspection. Addison-Wesley Publishing Company.

Good, P., 1994. Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses. Springer Verlag.

Graden, M., Horsley, P., and Pingel, T., 1986. The Effects of Software Inspections on a Major Telecommunications Project. AT&T Technical Journal, 65(3):32–40.

Greenwald, A., 1976. Within-Subjects Designs: To Use or Not to Use? Psychological Bulletin, 83(2).

Grizzle, J., 1965. The Two-period Chance-over Design and its Use in Clinical Trials. Biometrics, 21:314–320.

Hatton, L., 1998. Does OO Sync with How We Think? IEEE Software, 15(3):46-54.

Hayes, W., 1999. Research Synthesis in Software Engineering: A Case for Meta-Analysis. Proceedings of the International Symposium on Software Metrics, 143-151.

Hedges, L., and Olkin, I., 1985. Statistical Methods for Meta-Analysis. Academic Press.

Jacobson, I., 1992. Object-Oriented Software Engineering, Addison-Wesley.

Jacobson, I., Booch, G., Rumbaugh, J., 1998. The Unified Software Development Process, Addison-Wesley.

Johnson, P., and Tjahjono, D., 1998.Does Every Inspection Really Need a Meeting? Empirical Software Engineering, 3:9-35.

Keren, G., 1993. A Handbook for Data Analysis in the Behavioural Sciences - Methodological Issues, Chapter 19: Between- or Within-Subjects Design: A Methodological Dilemma. Lawrence Erlbaum Associates.

Kitchenham, B., Linkman, S., and Law, S., 1994. Critical Review of Quantitative Assessment. Software Engineering Journal, pages 43–53.

Laitenberger, O., and Atkinson, C., 1999. Generalizing Perspective-based Inspection to handle Object-Oriented Development Artefacts, Proceedings of the 21st International Conference on Software Engineering, Los Angeles, USA.

Laitenberger, O., and DeBaud, J.-M., 2000. An Encompassing Life-cycle Centric Survey of Software Inspection. Journal of Systems and Software. 50(1): 5-31.

Laitenberger, O., El Emam, K., Harbich, T., 2000. An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-based Reading of Code Documents, Accepted for Publication in IEEE Transactions on Software Engineering. Also available as Report of the International Software Engineering Research Network, ISERN-99-01.

Land, L., Sauer, C., and Jeffery, R., 1997. Validating the Defect Detection Performance Advantage of Group Designs for Software Reviews: Report of a Laboratory Experiment Using Program Code. In 6th European Software Engineering Conference, pages 294–309. Lecture Notes in Computer Science No 1301, ed. Mehdi Jazayeri, Helmut Schauer.

Linger, R., Mills, H., and Witt, B., 1979. Structured Programming: Theory and Practice. Addison-Wesley Publishing Company.

Lipsey, M., 1990. Design Sensitivity. Sage Publications.

McCall, J., 1994. Quality Factors. In J. Marciniak, editor, Encyclopedia of Software Engineering, volume 2, pages 958– 969. John Wiley and Sons.

Mehta, C., Patel, N., 1999. Exact Permutational Inference for Categorical and Nonparametric Data. In: Statistical Strategies For Small Sample Research. Ed: R. Hoyle. Sage Publications.

Miller, J., Wood, M., and Roper, M., 1998. Further Experiences with Scenarios and Checklists. Empirical Software Engineering, 3(1):37–64.

National Aeronautics and Space Administration. Software Formal Inspection Guidebook, 1993. Technical Report NASA-GB-A302, National Aeronautics and Space Administration, http://satc.gsfc.nasa.gov/fi/fipage.html.

Parnas, D., and Weiss, D., 1987. Active Design Reviews: Principles and Practice. Journal of Systems and Software, 7:259–265.

Porter, A., Votta, L., and Basili, V., 1995. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. IEEE Transactions on Software Engineering, 21(6):563–575.

Porter, A., and Votta, L., 1998. Comparing Detection Methods for Software Requirements Inspections: A Replication Using Professional Subjects. Empirical Software Engineering, 3:355-379.

Rifkin, S., and Deimel, L., 1994. Applying Program Comprehension Techniques to Improve Inspection, Proceedings of the 19th Annual NASA Software Engineering Workshop, NASA.

Rumbaugh, J., Blaha, M., Prembalini, W., Eddy, F., and Lorensen, W., 1991. Object-Oriented Modeling and Design, Prentice Hall.

Sandahl, S., Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M., and Ohlsson, N., 1998. An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections. Empirical Software Engineering, 3:327-254.

Travassos, G., Shull, F., Fredericks, M., and Basili, V., 1999. Detecting defects in object oriented designs: Using reading techniques to increase software quality. In Conference on Object-oriented Programming Systems, Languages & Applications (OOPSLA).

van Dijk, T., and Kintsch, W., 1984. Strategies of Discourse Comprehension. Academic Press, Orlando.

Votta, L., 1993. Does Every Inspection Need a Meeting? ACM Software Engineering Notes, 18(5):107–114.

Winer, B., Brown, D., and Michels, K., 1991. Statistical Principles in Experimental Design, 3rd edition. McGraw Hill Series in Psychology.

8 Appendix

8.1 Checklist

There are three basic criteria for evaluating the quality of UML-design diagrams: Correctness, Completeness, and Consistency. Each one of these criteria can be operationalized as follows:

Correctness
A diagram or a set of diagrams is correct if it is judged to be equivalent to some reference standard that is assumed to be an infallible source of truth. In the case of design diagrams, the analysis diagrams can be considered as such.

Completeness
A diagram or a set of diagrams is complete if no required elements are missing. It is judged by determining if the entities in the (set of) diagrams describe the aspects of knowledge being diagramed in sufficient detail for the part of the system under development.

Consistency
A diagram or a set of diagrams is consistent if there are no contradictions among its elements. This may be judged by checking the consistency between the elements of a single diagram or between several diagrams/ diagram types

Item no.	Where to look	How to detect
1	**All Design Diagrams**	Is each name unique?
2		Are all names used in the diagrams consistent?
3		Are all names used in the diagrams correct?
4	**Collaboration Diagrams**	Are the parameters and parameter types correct when compared to the “reads” and “changes” clause in the operation schemata?
5		Are all the input parameters supplied?
6		Are the messages correct when compared to the “sends” clause in the operation schema?
7		Is the message sequencing correct?
8		Do the effects of each collaboration diagram satisfy the specification given in the corresponding operation schema?
9		Is the collaboration diagram and its description in the pseudocode consistent?
10	**Design Class Diagrams**	Have all the classes been adequately defined and described?
11		Does each class attribute in the design class diagram have an associated data type?
12		Are all data types primitives?
13		Is the cardinality and arity of each association correct?
14		Are the design class diagrams correct when compared to the system class diagrams?
15		Are the design class diagrams consistent?
16		Do the collaboration diagrams behave correctly for all possible sets of parameter input values?
17	**Collaboration and Design Class Diagrams**	Does each class in the design class diagrams have at least one instance in a collaboration diagram?
18		Are all the class instances in the collaboration diagrams, their parameters, and their parameter types consistent with the design class diagrams and vice versa?

8.2 PBR Scenarios

8.2.1 The Designer Scenario

Assume you are inspecting the design diagrams from the perspective of a designer. The main concern of a designer is to ensure the correctness and completeness of the design diagrams with respect to the analysis diagrams. Correctness and completeness can be described as follows:
Correctness means that a diagram or a set of diagrams is judged to be equivalent to some reference standard that is assumed to be an infallible source of truth. In the case of design diagrams, the analysis diagrams can be considered as such.
Completeness means that a diagram or a set of diagrams includes all its required elements. It is judged by determining if the entities in the (set of) diagrams describe the aspects of knowledge being depicted in sufficient detail for the part of the system under development.
The development products , which are of relevance are the system class diagram, the operation schemata, the collaboration diagrams, and the design class diagrams.
Follow the instructions below and answer the questions carefully. Read the introductory part of the case study document and the use cases.
Continue with the following instructions:

- ➢ Locate the design class diagrams in the case study document. The design class diagrams need to be compared to the system class diagrams in the following manner:
  For each system class diagram ensure that each construct in the system class diagram is realized in the design class diagram. “Realizing” refers to
  - ➢ classes: Go to a class in the system class diagram and check that the class is also part of the design class diagram. Check the correctness of the class names, attribute names, and method names between the class in the system class diagram and the design class diagram.
  - ➢ Attributes: Check the correctness of the number and types of attributes between the classes in the system class diagram and design class diagram.
  - ➢ Methods: Check the correspondence of methods between the system class diagram and the design class diagram.
  - ➢ Associations: Check the correctness of the binary associations between classes in the system class diagram and the design class diagram. Compare the cardinality and the arity of the associations.
  - ➢ Constraints: Check the correctness of the constraints between classes in the system class diagram and the design class diagram.
  - ➢ Abstract Concepts: Check that abstract concepts such as association classes and ternary associations in the system class diagram have been properly mapped to constructs in the design class diagram.

  Be aware that the design class diagrams are a refinement of the system class diagrams and may therefore contain more classes than the system class diagrams!
  Mark the classes in the system class diagram with a pen after you have performed the checks.
  Look at the system class diagrams and check that each class has been marked in the previous effort.
- ➢ Locate the collaboration diagrams in the case study document. Each collaboration diagram needs to be compared to the corresponding operation schemata in the following manner:
  Check that the collaboration diagram has a message, which corresponds to the system operation.
  Find the entities in the operation schema preceded by the keyword supplied. These represent parameters of the operation. Find the message in the collaboration diagram corresponding to the system operation and check that the parameters and their types match. Mark the checked messages with a pen.
  For each entity in the “reads” and “change” clause not preceded by the keyword supplied, locate one or more corresponding messages in the collaboration diagram and check that it has the necessary parameters or return values to carry the required information.
  For each message in the “sends” clause check that there is a corresponding message in the collaboration diagram with corresponding parameters.
  Check that the messages in the collaboration diagram as a whole achieve the effects defined in the “result” clause of the operation schema.

While completing the instructions, answer the following questions:

1. Is there anything you realized in the analysis document that is not reflected in the design document?
2. Is an appropriate message sent to the corresponding object in the collaboration diagram for every attribute, object, or link that is changed in the operation schema?
3. Are the initial conditions for starting up a system operation clear and correct?
4. For every message that is defined in the "sends" clause of the operation schema, is there a corresponding message sent in the collaboration diagram?
5. Is the sequence of messages in the collaboration diagram correct?
6. For every attribute, object, or link that is changed in the operation schema, is an appropriate message sent to the corresponding object in the collaboration diagram?
7. Are there any discrepancies between the functionality defined in the "results" clause of the operation schemata and the one described in the collaboration diagram?

8.2.2 The Tester Scenario

Assume you are inspecting the design diagrams from the perspective of a tester. The main goal of a tester is to ensure the testability of the system design for performing the system test. High quality thus corresponds to full testability.

Hence, you will need to analyze test cases for the design diagrams, so if they are not available, arrange for them to be created. A test case consists of a set of input values plus a set of output values and/or state changes expected for each combination of values.

Follow the instructions below and answer the questions carefully. Read the introductory part and the use cases of the case study document.

Continue with the following instructions:

Locate the operation schema in the case study document. A test case is a set of values for the entities in the "reads" clause. Identify a set of test cases, which give rise to every possible outcome defined in the result clause of the operation schema. Document the values on a separate form.

Locate the collaboration diagram that corresponds to the operation schema. Walk through each collaboration diagram with these values as input values to exercise the described behavior.
Check that every test case yields the expected results.
Check that there are no branches and paths in the collaboration diagram that are not executed.
Check that the resulting behavior is consistent with the one described in the "results" clause of the operation schema.

Locate the design class diagrams in the case study document. check that your defined values are consistent with the attributes and attribute types of the design class diagram.

While completing the instructions, answer the following questions:

1. Do the branches in the collaboration diagram match the condition outcomes in the operation schema?
2. Are all possible sets of input values properly addressed by the operation schema and the pseudocode description?
3. Are the effects of a system operation specified under all possible circumstances?
4. Do the effects of each collaboration diagram satisfy the specification given in the operation schema for a corresponding system operation?

8.2.3 The Implementor Scenario

Assume you are inspecting a design diagram from the perspective of an implementor. The main task of an implementor is to transfer the design into code. High quality is determined by consistency and completeness of the design.

Consistency means that there are no contradictions among the elements of a diagram or a set of diagrams. This may be judged by checking the consistency between the elements of a single diagram or between several diagrams/diagram types.

Completeness means that a diagram or a set of diagrams includes all its required elements. It is judged by determining if the entities in the (set of) diagrams describe the aspects of knowledge being modeled in sufficient detail for the part of the system under development that it can be implemented.

Follow the instructions below and answer the questions carefully.

Read the introductory part and the use cases of the case study document.

Continue with the following instructions:

Locate the collaboration diagrams in the case study document. The collaboration diagrams need to be checked for consistency and completeness in the following manner:
- For each collaboration diagram ensure that each parameter has an associated type.
- For each message received by an object check that the class in the design class diagram has an associated method. Mark the class and method with a pen.
- Check that classes or methods in the design class diagram received no marks.
- Compare the collaboration diagram to its pseudo-code description and determine whether the pseudo-code description reflects the effects of the corresponding collaboration diagram.

Locate the design class diagrams in the case study document. The design class diagrams need to be checked for consistency and completeness in the following manner:
- For each class check that the attribute types are specified.
- For each class check that the methods for the class are defined.

While completing the instructions, answer the following questions:

1. Is there anything that prevents you from implementing the system design?
2. Are there any discrepancies in the algorithm defined in the collaboration diagram and the pseudocode description?
3. Is there anything in the collaboration diagrams or system class diagram that cannot be implemented?

Oliver Laitenberger is currently a researcher and consultant at the the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern. His main interests are software quality assurance with software inspections, inspection measurement, and inspection improvement. As a researcher, Oliver Laitenberger has been working for several years in the development and evaluation of inspection technology. As a consultant, he has worked with several international companies in introducing and improving inspections. Oliver Laitenberger received the degree Diplom-Informatiker (M.S.) in computer science and economics from the University of Kaiserslautern, Germany, in 1996.

Colin Atkinson is a Professor of Software Engineering at the University of Kaiserslautern, and a project/group leader at the associated Fraunhofer Institute for Experimental Software Engineering (IESE). His interests are centered on object and component technology and their use in the systematic development of software systems. He received his Ph.D. and M.Sc. in computer science from Imperial College, London, in 1990 and 1985, respectively, and his B.Sc. in Mathematical Physics from the University of Nottingham in 1983.

Maud Schlich is working for the Competence Center for Software Technologies and Continuous Education - a part of FhG IESE focusing on the transfer of base practices. Her main interests are inspections and testing, in particular for object-oriented software development. She is continuously active in teaching and has led numerous improvement projects, especially in small and medium enterprises.

Khaled El Emam is currently at the National Research Council in Ottawa. He is the current editor of the IEEE TCSE Software Process Newsletter, the current International Trials Coordinator for the SPICE Trials, which is empirically evaluating the emerging ISO/IEC 15504 International Standard world wide, and co-editor of ISO’s project to develop an international standard defining the software measurement process. Previously, he worked on both small and large research and development projects for organizations such as Toshiba International Company, Yokogawa Electric, and Honeywell Control Systems. Khaled El Emam obtained his Ph.D. from the Department of Electrical and Electronics Engineering, King’s College, the University of London (UK) in 1994. He was previously the head of the Quantitative Methods Group at the Fraunhofer Institute for Experimental Software Engineering in Germany, a research scientist at the Centre de recherche informatique de Montreal (CRIM), and a research assistant in the Software Engineering Laboratory at McGill University.