National Research Council Canada
Conseil national de recherches Canada
Institute for Information Technology
Institut de Technologie de l'information
ERB-1059

NRC·CNRC

An Internally Replicated Quasi-experimental Comparison of Checklist and Perspective-based Reading of Code Documents

Oliver Laitenberger, Khaled El Emam,
and Thomas Harbich
September 1999

Canada
NRC 43603

National Research Council Canada
Institute for Information Technology

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

An Internally Replicated Quasi-experimental Comparison of Checklist and Perspective-based Reading of Code Documents

Oliver Laitenberger, Khaled El Emam,
and Thomas Harbich
September 1999

An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-based Reading of Code Documents

Oliver Laitenberger
Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6
D-67661 Kaiserslautern
Germany
laiten@iese.fhg.de

Khaled El Emam[1]
National Research Council, Canada
Institute for Information Technology
Building M-50, Montreal Road
Ottawa, Ontario
Canada K1A OR6
Khaled.El-Emam@iit.nrc.ca

Thomas Harbich
Bosch Telecom GmbH
Gerberstrasse 33
D-71520 Backnang
Germany
Thomas.Harbich@pcm.bosch.de

Abstract

The basic premise of software inspections is that they detect and remove defects before they propagate to subsequent development phases where their detection and correction cost escalates. To exploit their full potential, software inspections must call for a close and strict examination of the inspected artefact. For this, reading techniques for defect detection may be helpful since these techniques tell inspection participants what to look for and, more importantly, how to scrutinise a software artefact in a systematic manner. Recent research efforts investigated the benefits of scenario-based reading techniques. 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 experimentally compare one scenario-based reading technique, namely perspective-based reading (PBR), for defect detection in code documents with the more traditional CBR approach. The comparison was performed in a series of three studies, as a quasi-experiment and two internal replications, with a total of 60 professional software developers at Bosch Telecom GmbH. Meta-analytic techniques were applied to analyse the data. Our results indicate that PBR is more effective than CBR (i.e., it resulted in inspection teams detecting more unique defects than CBR), and that the cost of defect detection using PBR is significantly lower than CBR. This study therefore provides evidence demonstrating the efficacy of PBR scenarios for code documents in an industrial setting.

Keywords: Software Inspection, Perspective-based Reading, Quasi-experiment, Replication, Meta-analysis

1 Introduction

Since Fagan's initial work presented in 1976 [24] 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% and 90% of the defects in a software artifact [25][30]. 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[2] can be reduced considerably. For example, a Monte Carlo simulation has indicated that on average, the implementation of code inspections reduces life cycle defect detection

[1] This work was partially done while El Emam was at the Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany.

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

costs by 39%, and that the implementation of design inspection reduces life cycle defect detection costs by 44% [5][3].

A software inspection usually consists of several activities including planning, defect detection, defect collection, and defect correction [53][4]. Inspection planning is performed by an organiser who schedules all subsequent inspection activities. The defect detection and defect collection activities 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 [55][85][44]. Therefore, defect detection can be considered 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 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[5].

In practice, most industrial inspection implementations use ad-hoc or checklist-based reading (CBR) during defect detection [24][30]. 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 [30]. 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 derives from the fact that a checklist is often based upon past defect information [10]. 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 inspected document for possible defects, which may cause him or her to get swamped with many unnecessary details [65].

A more recent approach, scenario-based reading [3], 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 artefact, as well as *what* that information should look like.

---

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

4 In this article, we model the inspection process in terms of its main activities. This allows us to be independent from a specific inspection implementation, such as the Fagan [24] or the Gilb [30] one.

[5] Other types of support can also be effective. For example, training sessions in program comprehension as presented in [71] can be beneficial to maximise the number of detected defects in code inspection.

Empirical evidence suggests that scenario-based reading techniques are valuable for defect detection in requirements documents [2][67][68]. However, no systematic empirical investigation has been performed to evaluate these reading techniques in the context of code inspections. This is of particular practical importance since a recent literature survey found that in many companies the use of software inspection for code artefacts predominates [53]. Moreover, existing empirical evaluations have focused mainly on the *effectiveness* of a scenario-based reading technique. They often provide very little evidence nor discussion on the *cost* of using a particular technique. However, the knowledge about the cost is a major factor influencing the adoption of a technique in practice.

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 code documents, we present a quasi-experiment and two internal replications conducted with 60 professional programmers at Bosch Telecom GmbH, Germany. During these studies, the effectiveness[6] of PBR and its cost per defect ratio are compared with those of CBR. Briefly, our results indicate that inspection teams applying PBR have a higher effectiveness and a lower cost per defect ratio than those applying CBR. From a methodological point of view, we found quasi-experimentation beneficial since it provides a path to follow for performing more empirical studies within industrial settings.

The 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 quasi-experiment and its two replications. 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 inspection as well as the hypotheses that we test in our experiment. An explanation of the mechanisms underlying these hypotheses is presented in Appendix A (see Section 9).

2.1 Evaluating Software Inspections

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 [5]. It is therefore important to maximise the number of defects detected through inspections, and to minimise 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 inspections. Cost is defined in terms of the effort to find 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: individual defect detection and defect collection in a meeting.

2.2 Two Inspector Inspections

Our focus is two inspector inspections. This means that two persons independently scrutinized the software artifact for defects before the inspection meeting takes place. This does not necessarily imply a limit to the overall inspection team size since other people may be involved in the inspection process,

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

[7] Another evaluative criterion of software inspections is their interval (i.e., calendar time elapsed) [85]. However, this is not addressed in the current study.

such as an independent moderator. However, the quality of the inspection process as well as the quality of the artifact after inspection is primarily determined by those people who scrutinize the artifact for defects (i.e., the inspectors).

Involving only two inspectors is in line with suggestions in Fagan's original work on software inspection [24]. He states that four people (i.e., the inspection moderator, the author, and two inspectors) constitute a good-sized inspection team, although circumstances may dictate otherwise. Such circumstances may be, for example, that a requirements document is inspected instead of a code artifact. Since the requirements cannot be checked against a preceding specification, a requirements inspection often involves more inspectors than other inspection types [27]. Furthermore, inspections are sometimes performed to promote a team spirit, transfer of skills and facilitate on-the-job training [22]. In such cases it may be desirable to have a larger inspection team.

There is already some quantitative evidence from academic environments that a two-person inspection team decreases inspection cost while maintaining inspection effectiveness [4][51]. For code inspections empirical evidence suggests that adding inspectors does not necessarily pay-off in terms of more detected defects. In a controlled experiment, Porter et al. [66] investigated 1, 2, and 4 inspector inspections. They found little difference in the inspection effectiveness of 2 and 4 inspectors. However, both were significantly more effective than 1 inspector inspections. Further, in practice, it is common that inspections are performed by a small team [24][30].

2.3 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 do [23][56]. 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* [3] extend the work of Parnas and Weiss on *Active Design Reviews* [65] and allocate specific responsibilities to inspectors. In addition, scenario-based reading techniques provide guidance for inspectors in the form of so-called scenarios on what to check and how to perform the required checks. A scenario typically consists of a limited, specific set of questions and a detailed set of instructions for an inspector about how to perform the checking.

Thus far, researchers have suggested three different scenario-based reading techniques: *Defect-based Reading* [67], a scenario-based reading technique based on function points [9], and *Perspective-based Reading* [2].

Porter et al. [67] describe defect-based reading scenarios. 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 [67][60], and similarly with professional subjects [68].[8]

Cheng and Jeffery [9] 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

[8] Two other student experiments did not confirm the superior defect detection capability of defect-based reading [28][75]. 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 5 cited DBR experiments [36] 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.

following the function-point scenarios did. However, the authors assert that experience was a confounding factor that biased the results of this experiment.

Basili et al. [2] 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 [24] reports that a piece of code should be inspected by its real tester, while Fowler [24] suggests that each inspection participant should take a particular point of view when examining the work product. Finally, Graden et al. [32] 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 from the inspected document and questions to analyse the extracted information. In the context of a controlled experiment at NASA [2], the researchers 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 have been 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.

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 requirements documents. Given that most companies focus their inspection activities on code documents (see the literature review in [53]), it would be of practical importance to evaluate these reading techniques for inspecting code documents as well.

In contrast to the other two scenario-based reading techniques, the PBR technique has been tailored for defect detection in code artefacts and applied in the context of a software development project at a German company [54]. However, a systematic comparison with CBR is still missing.

2.4 Experimental Hypotheses

Before we explain the expected effects when comparing PBR with CBR in the form of hypotheses, we define the following notation (we use d(a|b) for defining d(a) or d(b) respectively).

Term	Definition
e_i(CBR|PBR)	The individual defect detection effort for inspector i when using (CBR|PBR).
m (CBR|PBR)	The meeting effort after the individual inspectors used (CBR|PBR).
D (CBR|PBR)	The total number of unique defects found by two inspectors using (CBR|PBR)

Table 1: Notation.

We use this terminology to define our expectations. The rationale for these expectations is presented in Appendix A (see Section 9).

1. **The Effectiveness of PBR is Larger than the Effectiveness of CBR for Teams.**
   Given the anticipated defect detection benefits of PBR, then we would expect the following to hold within a given experiment:

$$D(CBR) < D(PBR) \qquad \textbf{Eqn. 1}$$

2. **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. Moreover, individual inspectors have to document some intermediate results, such as a call graph, a description of the function, or some test cases. It is therefore expected that an inspector will spend at least as much effort for defect detection using PBR than CBR:

$$e_i(PBR) \geq e_i(CBR) \qquad \textbf{Eqn. 2}$$

The effect that we anticipate is:

$$\frac{D(PBR)}{D(CBR)} > \frac{e_1(PBR) + e_2(PBR)}{e_1(CBR) + e_2(CBR)} \qquad \textbf{Eqn. 3}$$

which 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 counterpart 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:[9]

$$m(CBR) > m(PBR) \qquad \textbf{Eqn. 4}$$

It follows from Eqn. 1 and Eqn. 4:

$$\frac{m(CBR)}{D(CBR)} > \frac{m(PBR)}{D(PBR)} \qquad \textbf{Eqn. 5}$$

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, below we state our four null hypotheses:

[9] 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 code 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 abstractions that they have constructed during their individual reading, making it easier for an inspector to comprehend the reasoning behind the other perspective's defects. For CBR such abstractions are not available.

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

3.1 Implementation of Reading Techniques at Bosch Telecom GmbH

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. In our study, we provided the inspectors with a generic checklist. We limited the number of checklist items to 27 questions to fit on one page since this is recommended in the literature [10]. We structured the checklist according to the schema that is presented in [10]. 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 work product and the second component is a list of hints on how to identify a defect in the case of each problem spot. As problem spots we considered data usage (i.e., data declaration and data referencing), computation, comparison, control flow, interface, and memory. For each problem spot, we derived the checklist items from existing checklists such as [63], and books about the C Programming language [21][47]. Hence, the problem spots as well as the questions help reveal defects typical in the context of software development with the C programming language. Figure 1 presents an excerpt from the checklist.

Item no.	Where to look	How to detect
1	Data declaration	Are all variables declared before being used?
2		If variables are not declared in the code module, will it be sure that these variables are global ones?
3		Are the types of variables correct?
4	Data referencing	Are variables referenced that have no value (i.e., which have not been initialised)?
5		Are the indices of arrays within the specific boundaries?
6		Are all constants used correctly?

Figure 1: 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 [59], a software artefact needs to be inspected from each stakeholder's viewpoint. The basic goal of inspectors applying the PBR technique is therefore to examine the various documents of a software product from the perspectives of the product's various stakeholders for the purpose of identifying defects.

For probing 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 code of a function.

As depicted in Figure 2, 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 [2].

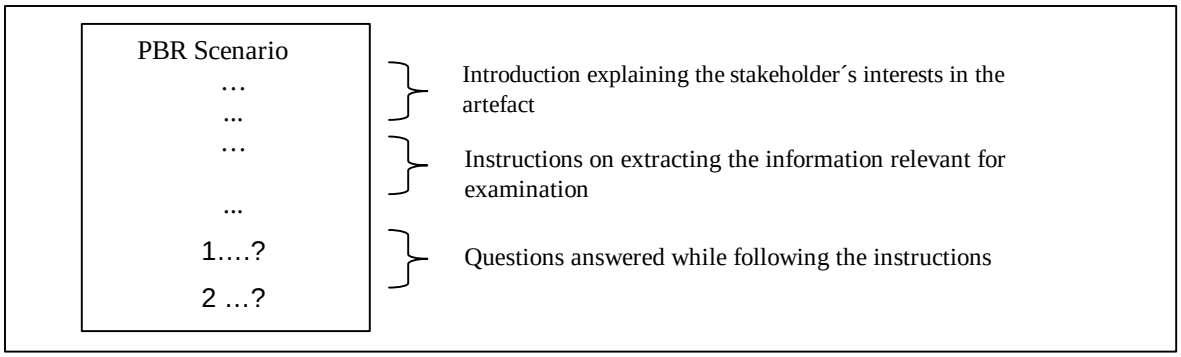

Figure 2: Content and Structure of a PBR Scenario.

The introduction describes a stakeholder's interest in the product 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 [84]. 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 to specific aspects of the artefact, which can be competently answered because of the attained understanding.

3.1.3 Development of a Scenario

Since the success of the PBR technique relies on the ability of software engineers not only to follow existing PBR scenarios but to create new ones, we briefly describe the process of how to derive the scenarios. This process is based on an article that describes how to tailor the PBR technique to the inspection of object-oriented development artefacts [52]:

1. The first process step is to identify the documents that contain pertinent information about a particular software product. The term "software product" refers not only to the final delivered software system but also to subsystems or even functions. It is possible to identify the documents of a particular product with the help of a product or process model since these models define the documents that must be created as part of the development process.
2. The second step is to specify the various stakeholders that have a vested interest in the product under inspection. As a starting point, the scenario developer may look at stakeholders that have a particular role in the software development process. These roles may be the producer of a preceding document (if existing), the producer of a subsequent document (if existing), the tester, and the

maintainer. The user of the product as well as domain experts may be helpful as well. Each of these represents a different (technical) perspective on the inspected product. If a particular document is not of interest to any stakeholder, its value to the overall software development process is questionable.

3. For each of the perspectives, the person that develops the scenario (i.e., the scenario developer) identifies what document and what kind of information in the document is most important for a particular stakeholder (e.g., to perform his or her role in the software development process), how to identify this kind of information, and how to extract it. For this, the scenario developer may interview the different stakeholders.
4. The scenario developer sets up the introductory part of the scenario by describing the interests of a stakeholder. Then, he or she develops instructions about how to identify and extract the required information. The granularity should have enough detail for an inspector to follow the given instructions step by step. Furthermore, it is important to somehow make inspectors document the extracted information (e.g., marking them with a coloured pen or writing parts of the information down). This captures what information the inspector has checked, for possible repetition at a later stage.
5. The fifth and final step in defining a scenario is to set up the questions an inspector is to answer based on the extracted information and the understanding of the product he or she has achieved. Characteristics of typical problems in a particular environment, exemplified by defect distributions, are useful information for developing the questions since they are often typical representatives of problems in an environment. However, only those questions are to be included in a scenario that an inspector can answer with the understanding he or she can achieve based on the extracted information.

This process describes in a generic manner how to identify perspectives and how to create an initial set of scenarios. The motivation for such a process comes from the need of practitioners to integrate new stakeholders or tailor the PBR technique to the inspection of different products. The scenarios crafted according to the process above are generic in the sense that they can be reused for the inspections of the same kind of document within or even across projects. In practice, scenarios are rarely if ever defined completely from scratch, but are typically adapted from previous scenarios based on the experience gained from applying them. In this sense, the scenarios in this paper represent a starting point for others to tailor them to their particular needs and environment.

3.1.4 Perspective-based Reading of Code Modules

In the context of our study, the products that are inspected are functions of a software system. The description of a function, that is, the physical inspected document, consists of its implementation in the C-programming language as well as of an operational description in the specification document.

Following the process that is outlined in [52], we identified two perspectives for the inspection of code documents at Bosch Telecom GmbH: A code analyst perspective and a tester perspective. In short, the code analyst's main interests are whether the code implements the right functionality while the tester checks whether the functionality was implemented right. For each perspective, we developed one scenario, which we provide in Appendix B and Appendix C according to the stated process. Both scenarios are generic, meaning that we did not tailor them in a particular manner to the application domain of Bosch Telecom GmbH.

An inspector reading the code document from a code analyst perspective identifies the different functions in the code module and extracts for each function a description of the functionality. This description can be compared to the specification and deviations are considered potential candidates for defects. For the extraction, he or she uses an abstraction procedure that is similar to the one suggested in Harlan Mill's reading technique "Reading by Stepwise Abstraction" [56].

An inspector reading the code module from the perspective of a tester identifies the different functions and tries to set up test cases with which he or she can ensure the correct behaviour of each function. Then, the inspector is supposed to mentally simulate each function using the test cases as input values and to compare the resulting output with the specification. Any deviation pinpoints potential defects.

Apart from the instructions that describe each activity in more detail, each scenario includes some questions that help focus the attention of inspectors on specific issues. In contrast to a checklist, the number of questions is limited and can be answered based on the results of the activities.

An inspection team consists of inspectors each of which has read the document from a different angle. The two perspective approach represents a minimal set of viewpoints with which we try to achieve high defect coverage. If some defects remain undetected we may include a third inspector that reads a code module from a different perspective. A primary candidate is the perspective of a maintainer. This derives from the fact that Votta reports that most of the issues found in a code inspection are so called soft maintenance issues [86]. Although these defects do not affect the functional behaviour of the software, their correction helps prevent code decay, which pays off later on.

3.2 Experimental Design

Our study was conducted as part of a large training course on software inspection within Bosch Telecom GmbH. We ran the initial quasi-experiment, and then replicated it twice (we refer to each one of these as a "run"). In each run we had 20 subjects, giving a total of 60 subjects who took part in the original quasi-experiment and its two replications. Each of the original quasi-experiment and its two replications took 6 days. Therefore the full study lasted 18 days.

In this section we describe the rationale behind the quasi-experimental design that we have employed, and the alternatives considered. In particular, since we make causal interpretations of the results, we discuss the potential weaknesses of a quasi-experiment and how we have addressed them.

To prelude the discussion of the experimental design, we emphasise that experimental design is, according to Hays [35], a problem in economics. Each choice that one makes for a design has its price. For example, the more treatments, subjects, and number of hypotheses one considers, the more costly an experiment is likely to be. This is particularly true for field experiments. Therefore, the objective is to choose a design that minimises the threats to validity within the prevailing cost constraints.

3.2.1 Description of the Environment

Bosch Telecom GmbH is a major player in the telecommunication market and develops high quality telecommunication systems (e.g., modern transmission systems based on SDH technology, access networks, switching systems) containing embedded software. One major task of the embedded software is the management of these systems. In the transmission systems and access network this comprises alarm management, configuration management, performance management, on-line software download, and on-line database down- and up- load. There are four typical characteristics for this kind of software. First, it is event triggered. Second, there are real time requirements. Third, the software must be highly reliable which basically means that the software system must be available 24 hours. Finally, the developed software system must be tailorable to different hardware configurations. Because of these characteristics and the ever increasing competition in the telecommunications market, high product quality is one of the most crucial demands for software development projects at Bosch Telecom GmbH.

Although reviews are performed at various stages of the development process to ensure that the quality goals are met, a large percentage of defects in the software system are actually found throughout the integration testing phase. A more detailed analysis of these defects revealed that many (typically one half) have their origin in the implemented code. Since defects detected in the integration testing phase are expensive (estimate: 5000 DM per detected and fixed defect), several departments at Bosch Telecom GmbH decided to introduce code inspections to detect these defects earlier and, thus, save detection as well as correction cost.[10]

[10] In this environment, code inspections are expected to reduce detection and correction costs for the following reasons. First, the integration test requires considerable effort just to set up the test environment. A particular test run consumes several hours or days. Once a failure is observed this process is stopped and the setup needs to be repeated after the correction. So, if some defects are removed beforehand, the effort for some of these cycles is saved (defect detection effort). Second, once a failure is observed it usually consumes considerable effort to locate or isolate the defect that led to the failure. In a code inspection, a defect can be located immediately. Hence, code inspection helps save correction cost (assuming that the effort for isolating defects is part of the defect correction effort).

Part of the effort to introduce code inspections was a code inspection training course for the software developers. We organised this as a quasi-experiment and two internal replications. This quasi-experimental organisation is advantageous for both participants and researchers. The participants not only learnt how software inspection works in theory but also practised them using a checklist as well as a PBR scenario for defect detection. In fact, the practical exercises offered the developers the possibility to convince themselves that software inspection helps improve the quality of code artefacts and, therefore, are beneficial in the context of their own software development projects. In a sense, these exercises help overcome the "not applicable here (NAH)" syndrome [43] often observed in software development organisations, and alleviated many objections against software inspections on the developers' behalf. From a researcher's perspective, the training effort offered the possibility to collect inspection data in a quasi-controlled manner.

3.2.2 Subjects

All subjects in our study were professional software developers of a particular business unit at Bosch Telecom GmbH. Each developer within this unit was to be trained in inspection and, thus, participated in this study. In order to capture their experiences we used a debriefing questionnaire. We captured the subjects' experience in the C-Programming language and in the application domain on a six-point scale as the most prevalent types of experience that may impact a subject's performance. Figure 3 shows boxplots of subjects' C-programming and application domain experiences.

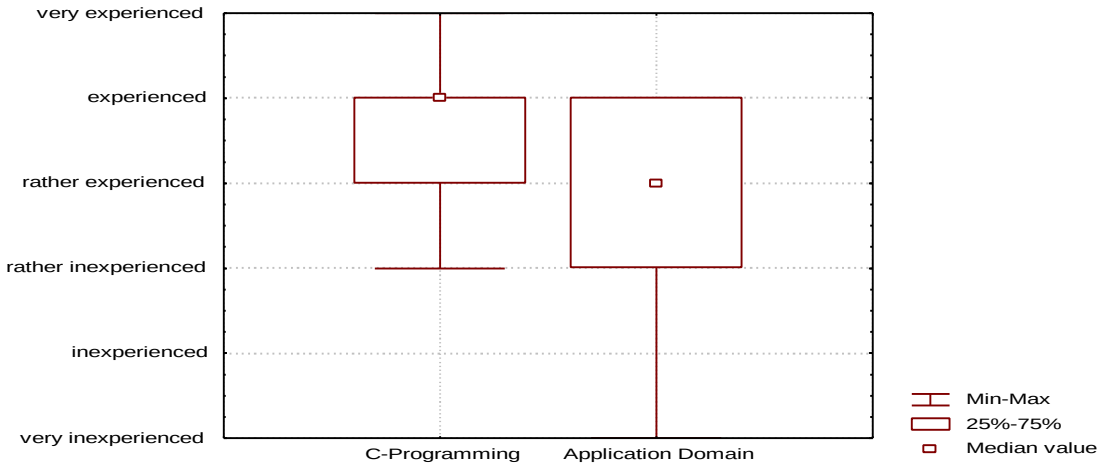

Figure 3 : Subjects' experience with the C-programming language and the application domain.

We found that subjects perceived themselves experienced with respect to the programming language (median of 5 on the 6 item scale) and rather experienced regarding software development in the application domain (median of 4 on the six item scale).

We can consider our subject pool a representative sample of the population of professional software developers. Because of cost and company constraints many empirical studies on software inspection were performed with students, that is, novices as subjects. Although these studies provide valuable insights, they are limited in the sense that the findings cannot be easily generalised to a broader context. As Curtis points out [18], generalisations of empirical results can only be made if the study population are professional software engineers rather than novices, and it has been forcefully emphasised that the characteristics of professional engineers differ substantially from student subjects [19]. The few existing results in the context of empirical software engineering support this statement since differences between professional software developers and novices were found to be qualitative as well as quantitative [89]. This means that expert and novice software developers have different problem solving processes causing experts not only to perform tasks better, but also to perform them in a different manner than novices. Hence, in the absence of a body of studies that find the same results with experts and novice subjects, such as the one presented by Porter and Votta [68], generalisations between the two groups are questionable and more studies with professional software developers as subjects are required.

3.2.3 Experimentation in an Industrial Setting

Experiments conducted in industrial settings with humans almost always have professional engineers as subjects. This is in contrast with "laboratory" experiments whereby university students are usually the subjects. However, working with professional engineers entails additional constraints on the design of an experiment. In our particular context, three constraints were important influences on the design:

Inability to Withhold Treatment. Since our study was performed in the context of training, 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[11]. 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 [46]. Previous empirical studies of different aspects of developer performance have found that individual performance differences can vary from 4:1 to 25:1 across experienced developers with equivalent backgrounds [7]. 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 control [7]. Repeated-measures designs enable a direct and unconfounded comparison between the different treatments [46]. 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 [57].

No Control Group. The validity of the concept of a "no-treatment" control group in software engineering research has been questioned [48]. 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 a previous study on reading techniques with professional engineers at NASA/GSFC [2]. In the context of our study the organisation did not perform any specific code reading prior to the training, only what can be characterised as informal individual desk checks. It would not have been possible to request that 60 engineers perform desk checks as part of the experiment due to the consequent substantial cost increase of the experiment. Therefore, we only compare the two reading techniques: CBR and PBR. This implies that it is impossible to determine whether either or both of the reading techniques are better than current practice. From a general scientific knowledge perspective this issue is not of concern since the questions being answered concern the relative performance of the two reading techniques. For the company that was involved, performance of the two reading techniques compared to the "usual" technique was not deemed sufficiently important for the required investment. There is justification for this position in that the literature has consistently demonstrated that formalised reviews, that is, software inspection, to be better than no reviews [5] or walkthroughs [62].

Natural Assemblage of Groups. For logistical reasons, training could only be done in groups of 10 subjects. Therefore, for each of the original quasi-experiment and the two replications we had two groups of 10 subjects each. This group size seemed reasonable in that it would have been difficult within the time frame of the study to schedule the same 3 consecutive days for more than 10 subjects. The individuals that formed each group were therefore not randomly assigned to groups, and multiple unknown selection factors outside our control determined the make up of each group. Certainly, availability and schedule conflicts played a role. Typically, random assignment can be regarded as a method of experimental control because it is assumed that over large numbers of subjects uncontrolled factors are distributed evenly over the treatment conditions [87]. Since random assignment of subjects to groups is the defining contrast between true experiments and quasi-experiments [16] or observational studies [12], this qualifies our study as a quasi-experiment. Quasi-experiments attempt to preserve as many of the properties of true experimentation as possible given the constraints of the industrial research setting [80][12].

---

[11] In a repeated-measures design, subjects receive two or more treatments.

[12] Campbell and Stanley point out that even in well-controlled true experiments, there are often nonrandom nuisance variables inherent to the experimental design that cannot be controlled [8]. Empirical support for this position can be found in [38].

3.2.4 Counterbalancing

A danger with repeated-measures designs is the existence of a carry-over effect [33]. A carry-over effect 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 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. In a previous experimental evaluation of reading techniques the authors noted that applying a prescriptive reading technique before a "usual" reading technique can lead to carry-over effects [2]. 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 as applicable, although one technique is more prescriptive than the other.

However, as will be noted in the post-hoc analysis section of this paper, the subjects perceived that CBR is easier to use than PBR. This means that if CBR is followed by PBR, the subjects may revert to the easier CBR when they are supposed to be using PBR, raising the danger of a carry-over effect. However, since PBR requires the construction of explicit abstractions, we can check that subjects in a CBR→PBR ordering actually use PBR by checking the abstractions that were constructed. In our study, no CBR→PBR contamination was found. If PBR is followed by CBR then there is likely to be less of a conformance problem since CBR is less prescriptive and the subjects found it easier.

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. The 2 groups in an experimental run were 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 persisted despite the counterbalancing.

3.2.5 The 2x2 Factorial 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 through n refer to the sequential order of implementing treatment or of recording observations within an experimental run. The horizontal sequence indicates the different treatments. Therefore, for the original quasi-experiment, the subjects were split in two groups. The first group (Group 1) performed a reading exercise using PBR first, and then measures were collected (O_1). Subsequently they performed a reading exercise using CBR, and again measures were collected (O_2). The second group (Group 2) performed the treatments the other way round. The vertical sequence indicates elapsed time during the whole study. Therefore, the first replication was executed with groups 3 and 4 after groups 1 and 2 and the second replication was run with groups 5 and 6 after groups 3 and 4.

Original Quasi-Experiment

Group 1	X_{PBR}/Module 1	O_1	X_{CBR}/Module 2	O_2
Group 2	X_{CBR}/Module 1	O_3	X_{PBR}/Module 2	O_4

First Replication

Group 3	X_{PBR}/Module 3	O_1	X_{CBR}/Module 4	O_2
Group 4	X_{CBR}/Module 3	O_3	X_{PBR}/Module 4	O_4

Second Replication

Group 5	X_{PBR}/Module 5	O_1	X_{CBR}/Module 6	O_2
Group 6	X_{CBR}/Module 5	O_3	X_{PBR}/Module 6	O_4

Table 2: Design of the Quasi-Experiment and its two Replications.

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

Different Code Modules Within a Run. Given that each group performs two reading exercises using different reading techniques, it is not possible for a group to read the same code modules 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 different code modules each time. Within a run, a module was used once with each reading technique so as not to confound the reading technique completely with the module that is used.

Different Code Modules Across Runs. During each of the quasi-experiment and its two replications different pairs of code modules were used. Since there was considerable elapsed time between each run, it was thought that past subjects may communicate with prospective subjects about the modules that were part of the training. By using different code modules, this problem is considerably alleviated.

Confounding. In this design the interaction between the module and the reading technique is confounded with the group main effect. This means that the interaction cannot be estimated separately from the group effect. However, it has been argued in the past that this interaction is not important because all modules come from the same application domain, which is the application domain that the subjects work in [2][83].

Effect of Intact Groups. As noted earlier, the groups in our study were intact (i.e., they were not formed randomly). The particular situation where there is an inability to assign subjects to groups randomly in a counterbalanced design was discussed by Campbell and Stanley [8]. In this design there is the danger that the group interaction with say practice effects confounds the treatment effect. However, if we only interpret a significant treatment effect as meaningful if it is not due to one group, then such a confounding would have to occur on different occasions in all groups in turn, which is a highly unlikely scenario [8]. Furthermore, it is noted that if there are sufficient intact groups and if they are assigned to the sequences at random, then the quasi-experiment would become a true experiment [8], which is also echoed by Spector [80]. This last point would argue for pooling the original quasi-experiment and its replications into one large experiment. However, it is known that pooling data from different studies can mask even strong effects [79][90], making it much preferable to combine the results through a meta-analysis [50]. A meta-analysis allows explicit testing of homogeneity of the groups. Homogeneity refers to the question whether the different groups share a common effect size. If homogeneity is ensured, one can combine the results to come up with an overall conclusion. Otherwise, the results must be treated separate from each other. An explicit test for homogeneity is conducted during our analysis to check if effect size estimates exhibit

greater variability than would be expected if their corresponding effect size parameters were identical [37].

Replication to Alleviate Low Power. Ideally, researchers should perform a power analysis[13] 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 CBR with PBR for code documents, and therefore it was not possible to use prior effect sizes as a basis for a power analysis. We therefore defined a medium effect size, that is an effect size of at least 0.5 [13],[14] as a reasonable expectation given the claimed potential benefits of PBR. Moreover, we set an alpha level of α=0.1. Usually, the commonly accepted practice is to set α=0.05[15]. However, in order to control Type I error (α) and Type II error (β) requires either rather large effects sizes or rather large sample sizes. This represents a dilemma in a software engineering context since much treatment effectiveness research in this area involves relatively modest effects sizes, and in general, small sample sizes. As pointed out in [57], if neither effect size nor sample size can be increased to maintain a low risk of error, the only remaining strategy – other than abandoning the research altogether – is to permit higher risk of error. This explains why we used a more relaxed alpha level for our studies..

With the anticipation of 10x2 inspector teams and an α=0.1, t-test power curves [49][57] for a one-tailed significance test[16] indicated that the experimental design has about a 0.3 probability of detecting, at least, a medium effect size. This was deemed to be a small probability of rejecting the null hypotheses if they were false (Cohen [13] recommends a value of 0.8). While this power level was not based on observed effect sizes, it already indicated potential problems in doing a single quasi-experiment without replications. After the performance of the quasi-experiment, we found for several hypotheses that the difference between CBR and PBR was not statistically significant. One potential reason for insignificant findings is low power. Using the *obtained* effect size from the quasi-experiment, an a posteriori power analysis was performed for all four hypotheses. Table 3 presents the a posteriori power levels.

---

[13] The power of a statistical test of a null hypothesis is the probability that it will lead to the rejection of the null hypothesis, i.e., the probability that it will result in the conclusion that the investigated phenomenon exists [13]. Statistical power analysis exploits the relationship among the following four variables involved in statistical inference:

- Power: The statistical power of a significance test is the probability of rejecting a false null-hypothesis.
- Significance level α: the risk of mistakenly rejecting the null-hypothesis and thus the risk of committing a Type I error.
- Sample size: the number of subjects/teams participating in the experiment.
- Effect size: the discrepancy between the null hypothesis and the alternate hypothesis. According to Cohen [13] effect size means "the degree to which the phenomenon is present in the population", or "the degree to which the null hypothesis is false. In a sense, the effect size is an indicator of the strength of the relationship under study. It takes the value zero when the null hypothesis is true and some other nonzero value when the null hypothesis is false. In our context, for example, an effect size of 0.5 between the defect detection effectiveness of CBR inspectors and PBR inspectors would indicate a difference of 0.5 standard deviation units.

Ideally, in planning a study, power analysis can be used to select a sample size that will ensure a specified degree of power to detect an effect of a particular size at some specified alpha level.

[14] In the allied discipline of MIS, Cohen's guidelines for interpreting effect sizes [13] have also been suggested in the context of meta-analysis [41].

[15] It is conventional to use an α level of 0.05. Cowles and Davis [17] trace this convention to the turn of the century, but credit Fisher for adopting this and setting the trend. However, some authors note that the choice of an α level is arbitrary [39][29], and it is clear that null hypothesis testing at fixed alpha levels is controversial [15][82]. More relevant for our endeavours, it is noted that typical statistical texts provide critical value tables for α=0.1 [77][78], indicating that this choice of α level is appropriate in some instances. As we explain in the text, our choice of α level was driven by power considerations.

[16] We used one-side hypothesis tests since we seek a directional difference between the two reading techniques.

	H_1	H_2	H_3	H_4
Quasi-Experiment	>0.9	0.49	>0.9	0.69
1st Replication	0.51	0.71	0.84	0.81
2nd Replication	0.84	0.28	0.76	0.49

Table 3: A Posteriori Power Analysis Results.

It is seen that low power was a potentially strong contributor for not finding statistical significance, making it difficult to interpret these results.[17] One possibility to tackle the problem of low power is to replicate an empirical study and merge the results of the studies using meta-analysis techniques. Meta-analysis techniques have been primarily designed to combine results from a series of studies, each of which had insufficient statistical power to reliably accept or reject the null hypothesis. This is the approach we have adopted and hence the prevalent reason for performing the two replications.

Replications to Increase Generalisability of Results. Replication of experimental studies provides a basis for confirming the results of the original experiment [20]. However, replications can also be useful for generalising results. A framework that distinguishes between close and differentiated replications has been suggested to explain the benefits of replication in terms of generalising results [69]. Our two replications can be considered as close replications since they were performed by the same investigators, using the same design and reading artefacts, under the same conditions, within the same organisation, and during the same period of time. However, there were also some differences that facilitate generalisation of the results. First, the subjects were different. Therefore, if consistent results are obtained in the replications we can claim that the results hold across subjects at Bosch Telecom GmbH. Second, the modules were varied. Again, if consistent results are obtained then we can claim that they are applicable across different code modules at Bosch Telecom GmbH. By varying these two elements in the replications, one attempts to find out if the same results occur *despite* these differences [69].[18]

Process Conformance. It is plausible that subjects do not perform the CBR and PBR reading techniques but revert to their usual technique that they use in everyday practice. This may occur, for example, if they are faced with familiar documents (i.e., documents from their own application domain within their own organisation) [83]. This is particularly an issue given that the subjects are professional software engineers who *do* have everyday practices. As alluded to earlier, with PBR it is possible to check this explicitly by examining the intermediate artefacts that are turned in. We did, and determined that the subjects did perform PBR as defined. For CBR, it will be seen in the post-hoc analysis section of the paper that more subjects found CBR easier to use for defect detection than their current reading technique (ad-hoc); actually more than double. Given such a discrepancy, it is unlikely that the subjects will revert to a technique that is harder to use when offered CBR. Therefore, we expect process conformance when using CBR to also be high.[19] It has also been suggested that subjects, when faced with time pressure, may revert to using techniques that they are more familiar with rather than make the effort of using a new technique [83]. During the conduct of the quasi-experiment and its replications, there were no time limits. That is, the subjects were given as much time as required for defect detection using each of the reading techniques. Therefore, this would not have affected the application of the reading techniques. Furthermore, when conducting analyses using defect detection cost, time limits would not invalidate conclusions drawn since there was no artificial ceiling on effort.

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

---

17 Not finding a statistically significant result using a low power study does not actually tell us very much because the inability to reject the null hypothesis may be due to the low power. A high power study that does not reject the null hypothesis has more credibility in its findings.

18 As noted in [69], however, this entails a gamble since if the results do differ in the replications, then one would not know which of the two elements that were changed are the culprit.

19 In [83] it is suggested that subjects could be asked to cross-reference the defects that they find with the specific steps of the reading technique. This would allow more detailed checking of process conformance. However, this would have added to the effort of the subjects, and it is unknown in what ways this additional activity would affect the effort when using CBR and PBR. If such an effect does occur differently for CBR and PBR, then it would contaminate the effort data that we collected.

learning [83]. If subjects do not receive feedback then they are less likely to continue applying the practices they learned during the first treatment. However, we found it very important to provide feedback at the end of each run.

Trainer Effects. For the quasi-experiment and its two replications, the same trainer was used. This trainer had given the same course before to multiple organisations, and therefore the quasi-experiment was not the first time that this course was given by the same person. Consequently, we expect no differences across the studies due to different trainers, nor due to the trainer improving dramatically in delivering the material.[20]

3.2.6 An Alternative Design

In educational settings, it frequently occurs that treatments are assigned to intact classes, and where the unit of analysis is the individual student (or teams) within the classes. This situation is akin to our current study whereby we have intact groups. In such situations one can explicitly take into account the fact that subjects are nested within groups and to employ a hierarchical design, and hence pool all the data into one larger experiment [80]. One can proceed by first testing if there are significant differences amongst groups. If there is group equivalence within treatments, then one can analyse the data as if they were not nested. However, if there were group differences, the unit of analysis would have to be the group and the analysis performed on that basis. This creates a difficulty in that reverting to a group unit of analysis would substantially reduce the degrees of freedom in the analysis. Hence, it would be harder to find any statistically significant effects even if such differences existed. Given the potential for this considerable methodological disadvantage, we opted a priori not to pool the results and rather perform a meta-analysis.

3.3 Experimental Materials

In each group the subjects inspected two different code modules. Apart from a code module, an inspector received a specification document which can be regarded as a type of requirements document for the expected functionality and which is considered defect free. All code modules were part of running software systems of Bosch Telecom GmbH and, thus, can be considered almost defect free. Table 4 shows the size of a code module in Lines of Code (without blank lines), their average cyclomatic complexity using McCabe's complexity measure [58], and the number of defects we considered in the analysis.

	Size(LOC)	Average Cycl. Complexity	Number of defects
Code Module 1	375	6.67	8
Code Module 2	627	11.00	9
Code Module 3	666	3.20	10
Code Module 4	915	4.38	16
Code Module 5	375	3.29	11
Code Module 6	627	5.44	11

Table 4: Characteristics of Code Modules.

3.4 Measurement

3.4.1 Dependent Variables

In this quasi-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-person inspection team (without defects found in the meeting). As

[20] Although, of course, an experienced trainer is still expected to improve as more courses were given; this is inevitable. However, their impact would be minimal.

the different code modules included a different number of defects we had to normalise the detected number of defects. We did this by dividing the number of detected defects by the total number of defects that is known. Hence, the dependent variable "*team defect detection effectiveness*" can be defined in the following manner:

$$\text{Team defect detection effectiveness} = \frac{\text{Defects found by a two – person team}}{\text{Total number of defects in the code module}} \qquad \textbf{Eqn. 6}$$

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 a two-person inspection team. The second one relates the meeting cost to the number of defects found by a two-person inspection team. And the third relates the sum of the defect detection cost and the meeting cost to the number of defects found by a two-person inspection team. Hence, the three instances of the dependent variable "*cost per defect for teams*" can be defined in the following manner:

$$\text{Cost per defect for the defect detection phase} = \frac{\text{Defect detection effort of two subjects}}{\text{Defects found by a two - person team (without meeting gains)}} \qquad \textbf{Eqn. 7}$$

$$\text{Cost per defect for the meetinge phase} = \frac{\text{Meeting effort of two subjects}}{\text{Defects found by a two - person team (without meeting gains)}} \qquad \textbf{Eqn. 8}$$

$$\text{Cost per defect for the overall inspection} = \frac{\text{Detection effort + Meeting effort}}{\text{Defects found by a two - person team (without meeting gains)}} \qquad \textbf{Eqn. 9}$$

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

Effects of Seeding Defects. We asked either the author or a very experienced software developer (if the author participated in a run[21]) to inject defects into the code modules. Hence, one person injected defects for the code modules that were used in either the quasi-experiment or one of its replications. These defects should be typical for the ones that are usually detected in testing and should not be detectable automatically by compilers or other tools, such as lint. There might be little bias since one person may insert defects that are easier or more difficult to detect than the ones inserted by another person. This bias may explain differences in the defect detection effectiveness and the cost per defect across the quasi-experiment and its replications. However, we only evaluate the difference between the two reading techniques *within* a study. This bias, therefore, does not have an impact on the individual study results. Furthermore, the tests of homogeneity that were performed (see below) indicate that there were no differences in team performance across the three studies.

Defect Reporting. In some cases the subjects reported more defects on their defect report forms than were seeded in a code module. 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 reanalysed the defect report forms of all the remaining subjects. Whether a defect was a true defect was our decision (to some extent based on discussions with the author or the person who seeded defects).

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 team meetings may be distorted by meeting gains and losses, which 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.

[21] Authors participated in a run in two cases. For these two modules, the authors had not worked on them for at least one year. Furthermore, they did not know what defects were injected. Therefore, the authors' inclusion in the study would incur minimal contamination, if any.

3.4.2 Independent Variables

We controlled two independent variables in the quasi-experiment and its replications:

1. Reading technique (CBR versus PBR)

2. Order of reading (CBR $\longrightarrow$ PBR versus PBR $\longrightarrow$ CBR).

3.5 Execution

The quasi-experiment and its two replications were performed between March and July 1998. Each consisted of two sessions and each session lasted 2.5 days and was conducted in the following manner (see Table 5). On the first day of each session, we did an intensive exercise introducing the principles of software inspection. 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 a code module from Bosch Telecom GmbH. Then, the subjects used the explained reading approach for individual defect detection in a code module. Regarding the PBR technique, the subjects either used the code analyst scenario or the tester scenario but not both. While inspecting a code module, the subjects were asked to log all detected defects 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 how to perform inspection meetings. To provide the participants with more insight into an inspection meeting, we randomly assigned two subjects to an inspection team and let them perform two inspection meetings (one for each code module) in which they could discuss the defects found in the two reading exercises. Of course, we ensured that within each of these teams one participant read a code module from the perspective of a code analyst and one read the code module from the perspective of a tester. The inspection team was asked to log all defects upon which both agreed. We then did an initial analysis by checking all defects that were either reported from individual subjects or from the teams, against the known defect list and presented the results on the third day (half-day). We consider the presentation of initial results a very important issue because, first, it gives experimenters the chance to get quick feedback on the results and, second, the participants have the possibility to see and interpret their own data, which motivates further data collection. At the end of each training session, each participant was given a debriefing questionnaire in which we asked the participant about the effectiveness and efficiency of inspections and the subjects' opinion about the training session.

	Day 1		Day 2		Day 3
	Morning	Afternoon	Morning	Afternoon	Morning
Group x	Introduction of Inspection Principles	PBR Explanation Defect Detection with PBR (Module A)	CBR Explanation Defect Detection with CBR (Module B)	Team Meetings for Module A and Module B	Feedback on Results
Group y	Introduction of Inspection Principles	CBR Explanation Defect Detection with CBR (Module A)	PBR Explanation Defect Detection with PBR (Module B)	Team Meetings for Module A and Module B	Feedback on Results

Table 5: Execution of the Quasi-Experiment and its Replications.

3.6 Data Analysis Methods

3.6.1 Analysis Strategy

To understand how the different treatments affected individual and team results across the quasi-experiment and its replications, we started the data analysis by calculating some descriptive statistics of the individual and team results. We continued testing the stated hypotheses using a t-test for repeated measures, i.e., a matched-pair t-test [1]. The t-test allowed us to investigate whether a difference in the defect detection effectiveness or the cost per defect ratio is due to chance. Although the t-test is robust against violation of certain assumptions (i.e., the normality[22] and homogeneity of the data), we also performed the Wilcoxon signed ranks test [78], which is the non-parametric counterpart of the matched-pair t-test. The Wilcoxon signed rank test corroborated the findings of the t-test in all cases. Hence, we do not present the detailed results of this test. We run the statistical tests for the quasi-experiment and the two replications separately.

Our experimental design permits the possibility of carry-over effects. Grizzle [34] 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. Only if this test is not significant, even at a very relaxed level, is a further analysis of the total data set appropriate. In that case, all the obtained data may properly be used in separate tests of significance for treatments. The description of the specific approach for testing for carry-over effects is provided in Appendix D (see Section 12).

One goal of any empirical work is to produce a single reliable conclusion, which is, at first glance, difficult if the results of several studies are divergent or not statistically significant in each case. Close replication and replication in general, therefore, raises questions concerning how to combine the obtained results with each other and with the results of the original study. Meta-analysis techniques can be used to merge study results. Meta-analysis refers to the statistical analysis of a collection of analysis results from individual studies for the purpose of integrating the findings.

Although meta-analysis allows the combination and aggregation of scientific results, there has been some criticism about its use [31]. The main critical points include diversity, design, publication bias, and dependence of studies. Diversity refers to the fact that logical conclusions cannot be drawn by comparing and aggregating empirical results that include different measuring techniques, definitions of variables, and subjects because they are too dissimilar. Design means that results of a meta-analysis cannot be interpreted because results from "poorly" designed studies are included along with results from "good" designed studies. Publication bias refers to the fact that published research is biased in favour of significant findings because non-significant findings are rarely published. This in turn leads to biased meta-analysis results. Finally, dependence of the studies means that meta-analyses are conducted on large data sets in which multiple results are derived from the same study.

Diversity, design, and publication bias do not play a role in our case. However, we need to discuss the issue of study dependency in more detail. It has been remarked that experiments using common experimental materials exhibit strong inter-correlations regarding the variables involving the materials. Such correlations result in non-independent studies [61]. In general, Rosenthal [72] discusses issues of non-independence in meta-analysis for studies performed by the same laboratory or research group, which is our case since we have two internal replications. He presents an example from research on interpersonal expectancy effects demonstrating that combining all studies across laboratories and combining studies where the unit of analysis is the laboratory results in negligible differences. Hence, in practice, combining studies from the same laboratory or research group is not perceived to be problematic.

---

[22] Before starting the analysis, we performed the Shapiro-Wilks' W test of normality, which is the preferred test of normality because of its good power properties [81]. In all cases we could not reject the null hypothesis that our data were normally distributed.

Another legitimate question is whether it is appropriate to perform a meta-analysis with a small meta-sample (in our case a meta-sample of three studies). It has been noted that there is nothing that precludes the application of meta-analytic techniques on a small meta-sample [50] "Meta-analyses can be done with as few as two studies or with as many studies as are located. In general, the procedures are the same […] having only a few studies should not be of great concern". For instance, Kramer and Rosenthal [50] report on a meta-analysis of only *two* studies evaluating the efficacy of a vaccination to SIV in monkeys using data from Cohen [14]. In the realm of software engineering, meta-analyses have also tended to have small meta-samples. For example, Hayes [36] performs a meta-analysis of five experiments evaluating DBR techniques in the context of software inspections. Miller [61] performs a meta-analysis of four experiments comparing defect detection techniques.

3.6.2 Comparing and Combining Results using Meta-Analysis Techniques

Meta-analysis is a set of statistical procedures designed to accumulate empirical results across studies that address the same or a related set of research questions [88]. As pointed out by Rosenthal [72] there are two major ways to merge and subsequently evaluate empirical findings in meta-analysis - in terms of their statistical significance (e.g., p-levels) and in terms of their effect sizes (e.g., the difference between means divided by the common standard deviation). One reason for the importance of the effect size is that many statistical tests, such as the t-test, can be broken down mathematically in the following two components [74]:

Significance test = Effect Size x Size of study

This relationship reveals that the result of a significance test, such as the t-test, is determined by the effect size *and* the size of the study. Many researchers however only make decision based on the fact of whether the result of applying a particular test is statistically significant. They often do not take the effect size or the size of the study into consideration. This almost exclusive reliance of researchers on results of null hypothesis significance testing alone has been heavily criticised in other disciplines, such as psychology and the social sciences [15][73][76]. We therefore explicitly take the effect size into consideration during the meta-analysis.

Two major meta-analytic processes can be applied to the set of studies to be evaluated: Comparing and combining [73]. When studies are compared as to their significance levels or their effect sizes, we want to know whether they differ significantly among themselves with respect to significance levels or effect sizes, respectively. This is referred to as homogeneity. When studies are combined, we want to know how to estimate the overall level of significance and the average effect size, respectively. In most cases, researchers performing meta-analysis, first, compare the studies to determine their homogeneity. This is particularly important in a software engineering context since, there, empirical studies are often heterogeneous [6][60]. Once it is shown that the studies are, in fact, homogeneous the researchers continue with a combination of results. Otherwise, they look for reasons that cause variations. According to this procedure, we first compared and combined p-values and, second, compared and combined effect sizes of the team results. We limited the meta-analysis to the team results since these are of our primary interest.

Comparing and Combining p-Values

For comparing the p-values, we followed a procedure that is described by Rosenthal [73]: Given the three p-levels of our studies to compare, we find the standard normal deviate Z corresponding to each p-level. The Z's will have the same sign if all studies show effects in the same direction. They will have different signs if the results are not in the same direction. The statistical significance of the heterogeneity of the Z's (and the p's associated with the Z's) can be obtained by first calculating:

$$c^2 = \sum \left(Z_j - \overline{Z}\right)^2 \qquad \textbf{Eqn. 10}$$

which is distributed as c^2 with k-1 degrees of freedom [88]. In this equation Z_j is the Z for any one study and $\overline{Z}$ is the mean of all the Z's obtained. A significant c^2 tells us that the Z's we have tested for homogeneity differ significantly among themselves. Assuming we do have homogeneity, we can combine the p-values. Perhaps the most widely used meta-analysis procedure to combine p-values is that of Fisher [26]. It is a method to combine the statistical test results of different experiments that validated the same hypothesis. The Fisher combined test is based on the product of probabilities of the statistical tests. If the natural logarithms of these probabilities are calculated, multiplied by minus two (-2), and then summed, a statistic with degrees of freedom equal to two times the number of combined tests (2k) is obtained. The test procedure becomes:

Reject the null hypothesis for combined tests if

$$P = -2 \times \sum_{i=1}^{k} \ln p_i \geq C \qquad \textbf{Eqn. 11}$$

where the critical value C is obtained from the upper tail of the c^2-distribution with 2k degrees of freedom.

Comparing and Combining Effect Sizes

Before we can start to compare and combine the effect sizes of our quasi-experiment and our replications, we need to define a measure for the effect size. Which effect size measure to choose is unimportant since there are simple equations to transform one effect size measure to another [73]. In our case, we decided to use Hedges g [37] as our measure of effect size. Hedges g is defined as follows:

$$g = \frac{\overline{x}_{PBR} - \overline{x}_{CBR}}{s} \qquad \textbf{Eqn. 12}$$

where $\overline{x}_{PBR}$ and $\overline{x}_{CBR}$ are the respective treatment means and s is the pooled sample standard deviation:

$$s = \sqrt{\frac{s_{PBR}^{\ 2} + s_{CBR}^{\ 2}}{2}} \qquad \textbf{Eqn. 13}$$

Hedges and Olkin [37] report that g has a small sample bias. However, it is easy to remove this bias by defining an unbiased estimator d for the effect size:

$$d = \left(\frac{\overline{x}_{PBR} - \overline{x}_{CBR}}{s} \right) \times J(N-2) \qquad \textbf{Eqn. 14}$$

A table of J(N-2) is presented in [37]. We used a J(N-2)-value of 0.8882 in the calculations for the quasi-experiment and a J(N-2)-value of 0.9027 in the calculations for the two replications. Hedges and Olkin also present an approximation of J(N-2), which is defined as follows:

$$J(N-2) \cong \left(1 - \frac{3}{4N-9}\right) \qquad \textbf{Eqn. 15}$$

In addition to the small sample bias, one has to consider that our quasi-experiment is a "within-subject" design meaning that the two samples being compared are not independent of each other. As a consequence, the t-test is based on a sampling error estimate or "error term" that is reduced in proportion to the strength of the correlation $r_{PBR/CBR}$ between the paired values [57]. In fact, for paired samples the denominator of effect size is not s, the pooled standard deviation, but

$$s \times \sqrt{1 - r_{PBR/CBR}} \qquad \textbf{Eqn. 16}$$

This formula can be used to produce the effect size for paired observations as follows:

$$d = \left(\frac{\bar{x}_{PBR} - \bar{x}_{CBR}}{s \times \sqrt{1 - r_{PBR/CBR}}} \right) \times J(N-2) \qquad \textbf{Eqn. 17}$$

Before pooling the estimates of effect size from a series of k studies, it is important to determine whether the studies can reasonably be described as sharing a common effect size. A statistical test for the homogeneity of effect sizes is formally a test of the hypotheses that all the effect sizes of the studies are the same versus the alternative hypothesis that at least one of the effect sizes differs from the remainder.

Hedges and Olkin [37] present a test statistic *Q* for testing this hypothesis. The test statistic is defined in the following manner:

$$Q = \sum_{i=1}^{k} \frac{d^2_i}{\hat{s}^2(d_i)} - \left(\frac{\left(\sum_{i=1}^{k} \frac{d_i}{\hat{s}^2(d_i)} \right)^2}{\left(\sum_{i=1}^{k} \frac{1}{\hat{s}^2(d_i)} \right)} \right) \qquad \textbf{Eqn. 18}$$

with the estimated variance of d_i defined as:

$$\hat{s}^2(d_i) = \frac{8 + d^2_i}{4N} \qquad \textbf{Eqn. 19}$$

If all studies have the same population effect size (i.e., the null hypothesis is true) then the test statistic Q has an asymptotic $?^2$-distribution with k-1 degrees of freedom. If the value of *Q* exceeds the 100(1-α)-percent critical value of the $?^2$-distribution with k-1 degrees of freedom, we reject the hypothesis that the effect sizes are homogeneous and decide not to pool the effect size estimates. If we cannot reject this null hypothesis, we can pool the effect size by calculating the mean effect size.[23]

[23] It should be noted that previous meta-analyses in software engineering with equivalently small meta-samples did detect heterogeneity [36][61].

The final question that needs to be addressed is whether the effect size can be considered practically significant, that is, whether the effect is large enough to be of interest. Cohen [13] reported that across a wide sampling of behavioural science research, effects of around 0.8 were at the large end of the range of what has been found, 0.5 was about medium, and 0.2 was at the small end of the range. In the absence of adequate effect size ranges in a software engineering context, we consider effect sizes of $d \geq 0.5$ to be of practical significance.

4 Results

In this section we present the detailed results. Note that all t-values and effect sizes have been calculated to be consistent with the direction: $\overline{x}_{PBR} - \overline{x}_{CBR}$.

4.1 Defect Detection Effectiveness

Figure 4 shows boxplots of the teams' defect detection effectiveness.

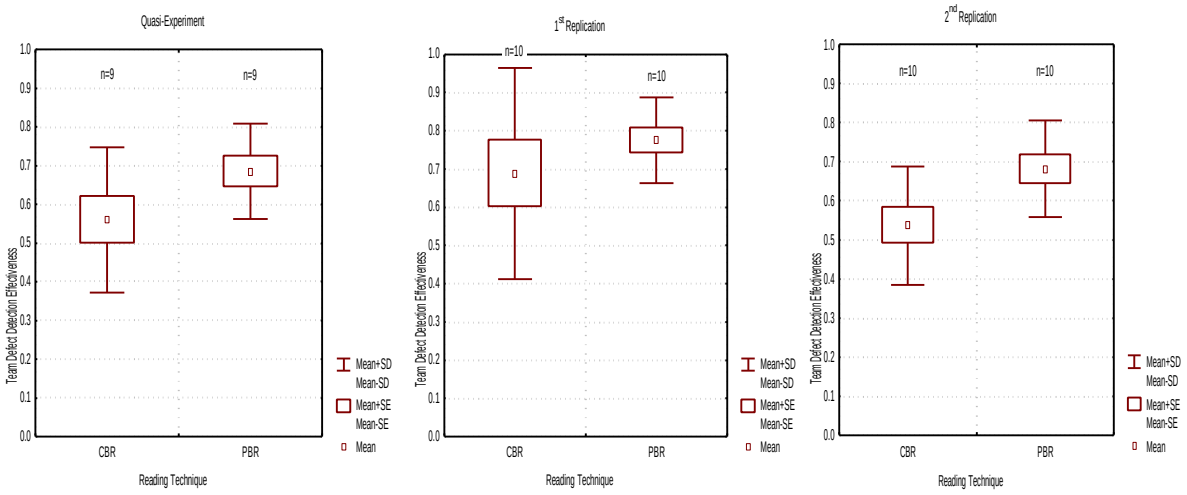

Figure 4: Box-Plots of the Team Defect Detection Effectiveness.[24]

The boxplots show that the inspection teams detected on average between 58% and 78% of the defects in a code module. These percentages are in line with the ones reported in the literature [30]. Teams using PBR for defect detection had a slightly higher team defect detection effectiveness than the same teams using CBR. In addition to the effectiveness difference, the boxplots illustrate that PBR teams exhibit less variability than CBR teams. The lower variability for PBR may be explained by the fact that the more prescriptive approach for defect detection to some extent removes the effects of human factors on the results. Hence, all the PBR teams achieved similar scores. However, before providing further explanation of these results, we need to check whether the difference between CBR and PBR is due to chance.

We first assessed whether there is a carry-over effect for the team effectiveness. The results, which we summarised in Appendix F, indicate no carry-over effect. Therefore we can proceed with the analysis of the data from the two periods.

We investigated whether the difference among teams is due to chance. Table 6 presents the summary of the results of the matched-pair t-test for the team defect detection effectiveness.

[24] For one team in the quasi-experiment, one member of one team dropped out of the study. Therefore we have only nine teams for the analysis.

	t-value	df	p-value (one-sided)
Quasi-Experiment	3.09	8	0.007
1st Replication	1.37	9	0.10
2nd Replication	2.39	9	0.02

Table 6: t-Test Results of the Team Defect Detection Effectiveness.

Taking an alpha level of $\alpha = 0.1$ we can reject H_{01} for the quasi-experiment and the 2nd replication. We cannot reject hypothesis H_{01} for the 1st replication. The findings suggest a treatment effect in two out of three cases, which is not due to chance.

At this point we need to decide whether there is an overall treatment effect across studies. Therefore, we performed a meta-analysis as described previously to compare and combine the results. We put the data for performing the calculations in Appendix E. The test for homogeneity for p-values results in p=0.7. Hence, we cannot reject the null hypothesis that our p-values are homogeneous. This means that we can combine the p-values of the three studies according to Fisher's procedure in the following manner:

$$P = -2 \times \sum_{i=1}^{k} \ln p_i = 22.14$$

Based on the c^2-distribution, this value of P results in a p-value of p=0.000016. Hence we can reject the null hypothesis and conclude that the resulting combination of the quasi-experiment and its replications revealed that a team using the PBR technique for defect detection had a significantly higher defect detection effectiveness than the team using CBR.

We continued the analysis by looking at the effect sizes. Table 7 reveals the effect sizes of the three studies.

	S pooled	Hedges g	d
Quasi-Experiment	0.16	0.79	1.46
1st Replication	0.21	0.42	0.81
2nd Replication	0.14	1.06	0.97

Table 7: Effect Sizes of the Team Defect Detection Effectiveness.

Table 7 shows that the 1st replication has the lowest effect size, which explains why the results of the test were not statistically significant. To compare and combine the effect sizes, we first checked the effect size homogeneity by calculating *Q*. The calculated value of *Q* is *Q* = 0.91 which leads to a p-value of p=0.63. Hence, we cannot reject the null hypothesis that the effect sizes are homogeneous. The combination of the effect sizes reveals a mean effect size value of 1.08. This represents a large effect size, i.e., one would really become aware of a difference in the team defect detection effectiveness between the use of PBR and CBR. Based on our findings, we therefore can reject hypothesis H_{01}.

For three of the modules in our study all defects were detected using PBR. For the other three all defects except one were detected. No discernible pattern in terms of the types of defects that were not detected could be identified.

4.2 Cost per Defect for the Defect Detection Phase

Before looking at the team results, we first investigated how much effort each subject consumed for defect detection using either of the techniques and whether there is a difference. Table 8 depicts the average effort in minutes that a subject spent for defect detection using either CBR or PBR and the results of the matched-pair t-test.

	CBR	PBR	p-values (one sided)
Quasi-Experiment	115.2	145.6	0.0003
1st Replication	118.8	132.5	0.16
2nd Replication	168	150.5	0.13

Table 8: Average Effort per Subject for the Defect Detection Phase.

We found that in the quasi-experiment and the 1st replication, the subjects consumed less effort for CBR than for PBR. However, only in the first case the difference was statistically significant. This finding seems to indicate that if there is a significant difference PBR seems to require more effort on the inspector's behalf. The question is whether the extra effort is justified in terms of more detected and documented defects in the team meeting.

Figure 5 depicts boxplots of the cost per defect ratio of both inspectors on an inspection team.

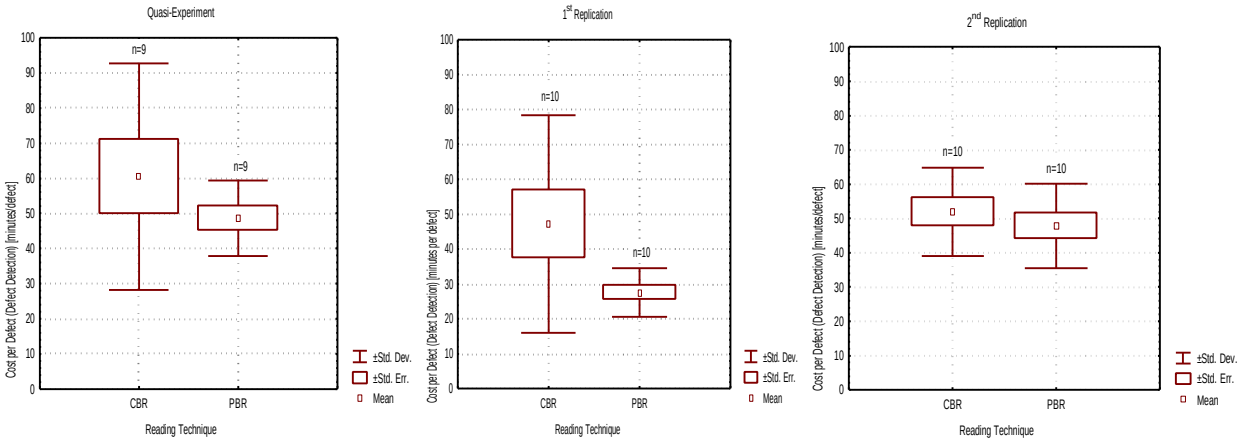

Figure 5: Box-Plots of the Cost per Defect Ratio for the Defect Detection Phase.

Figure 5 reveals that on average an inspection team consumed between 28 and 60.5 minutes per defect. The average cost per defect of PBR teams is consistently lower than the cost per defect of CBR teams. In this case, there is also less variability in the cost per defect ratio of PBR teams. Based on these findings, the extra effort for PBR, if any, seems to be justified because PBR teams have a better cost per defect ratio than CBR teams.

A formal test for carry-over effects was conducted, and none were identified. The test results are summarised in Appendix F.

	t-value	df	p-value (one-sided)
Quasi-Experiment	-1.33	8	0.11
1st Replication	-1.93	9	0.04
2nd Replication	-0.75	9	0.23

Table 9: t-Test Results of the Cost per Defect Ratio for the Defect Detection Phase.

Table 9 shows the t-test results for the cost per defect during the defect detection phase. Taking $\alpha = 0.1$ we can reject H_{02} for the 1st replication. We cannot reject hypothesis H_{02} for the quasi-experiment and the 2nd replication.

To compare and combine the results we first performed the homogeneity check for p-values. This value with 2 degrees of freedom results in a p-value of p=0.77. Hence, we cannot reject the hypothesis that our p-values are homogeneous. This finding allowed us to combine the p-values of the three studies. Calculating the combination of the p-values according to Fisher's procedure results in:

$$P = -2 \times \sum_{i=1}^{k} \ln p_i = 13.57$$

Based on the c^2-distribution, this value of P results in a p-value of p=0.001. Hence we can reject the hypothesis H_{02} and conclude that the resulting combination of the quasi-experiment and its replications revealed that a team using the PBR technique for defect detection had a significantly lower defect detection cost per defect than the team using CBR.

We continued the analysis by looking at the effect sizes. Table 10 reveals the effect sizes of the three studies.

	S pooled	Hedges g	d
Quasi-Experiment	24.03	-0.50	-0.72
1st Replication	22.59	-0.87	-0.78
2nd Replication	12.59	-0.33	-0.30

Table 10: Effect Sizes of the Cost per Defect Ratio for the Defect Detection Phase.

Table 10 shows that the quasi-experiment and 2nd replication have the lowest effect size, which explains why the results of these tests were not statistically significant. To compare and combine the effect sizes, we first checked the effect size homogeneity by calculating *Q*. The calculated value of *Q* is *Q* = 0.64 which leads to a p-value of p=0.73. Hence, we cannot reject the hypothesis that the effect sizes are homogenous. The combination of the effect sizes gives a mean effect size value of 0.6. Considering our effect size threshold of 0.5, we can conclude that we have, in fact, found an effect of practical significance.

We therefore can reject hypothesis H_{02}.

4.3 Cost per Defect for the Meeting Phase

We subsequently consider the cost per defect when accounting for the meeting phase. Figure 6 shows the boxplots for the quasi-experiment and its replications.

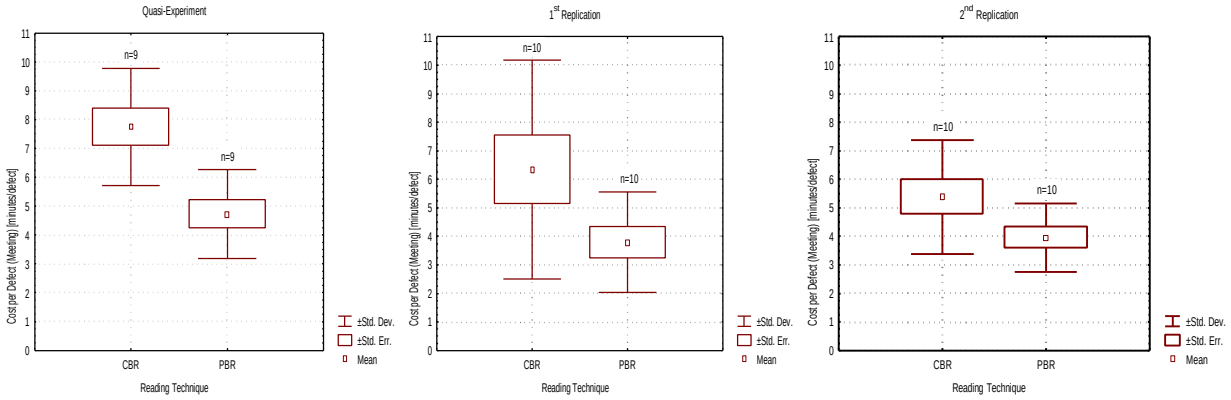

Figure 6: Box-Plots of the Cost per Defect for the Meeting Phase.

Figure 6 reveals that the average cost per defect ratio of PBR was lower than the CBR one when only considering the effort of the meeting phase. Although there seems to be less variability for the 1st replication, there does not seem to be as much differences in the variability for the quasi-experiment and the 2nd replication. Overall, this result indicates that the meeting cost per defect is higher for CBR than for PBR.

The results of a formal test for carry-over effects did not indicate any carry-over-effects, and is summarised in Appendix F.

	t-value	df	p-value (one-sided)
Quasi-Experiment	-5.20	8	0.0004
1st Replication	-2.05	9	0.035
2nd Replication	-2.55	9	0.016

Table 11: t-Test Results of the Cost per Defect Ratio for the Meeting Phase.

Taking $\alpha = 0.1$ we can reject H_{03} for all three studies. The findings suggest a treatment effect in all three cases, which is not due to chance.

We calculated $\sum (Z_j - \overline{Z})^2 = 1.396$. This value with 2 degrees of freedom results in a p-value of p=0.50. Hence, we cannot reject the null hypothesis that our p-values are homogeneous. Calculating the combination of the p-values according to Fisher's procedure results in:

$$P = -2 \times \sum_{i=1}^{k} \ln p_i = 30.59$$

Based on the c^2-distribution, this value of P results in a p-value of p<0.00001. Hence we can reject the hypothesis H_{03} and conclude that the resulting combination of the quasi-experiment and its replications revealed that a team using PBR for defect detection had a significantly lower cost per defect for the meeting phase than the team using CBR.

We continued the analysis by looking at the effect sizes. Table 12 reveals the effect sizes of the three studies.

	S pooled	Hedges g	d
Quasi-Experiment	1.80	-1.64	-2.17
1st Replication	2.99	-0.85	-1.06
2nd Replication	1.65	-0.87	-0.86

Table 12: Effect Sizes of the Cost per Defect Ratio for the Meeting Phase.

Table 12 shows that the 1st replication has the lowest effect size. However, the effect size is large enough for the test results to be statistically significant. To compare and combine the effect sizes, we first checked the effect size homogeneity by calculating *Q*. The calculated value of *Q* is *Q* = 3.23 which leads to a p-value of 0.20. Hence, we cannot reject the null hypothesis that the effect sizes are homogeneous. The combination of the effect sizes gives a mean effect size value of 1.36. This represents a large effect size considering our effect size threshold of 0.5.

It was postulated that the increased effort and, thus, the higher cost per defect ratio for subjects using PBR lead to an increased understanding of the documents. We therefore investigated the subjects´ perceptions of understanding the documents using both reading techniques. The debriefing questionnaire contained a question asking the subjects how well they understood the inspected code artefact using

either CBR or PBR. Figure 7 presents a histogram of the results of this question from the quasi-experiment and its replications. There is a clear trend confirming our expectation that using a PBR scenario for defect detection improves a subject's understanding of the inspected code artefact.

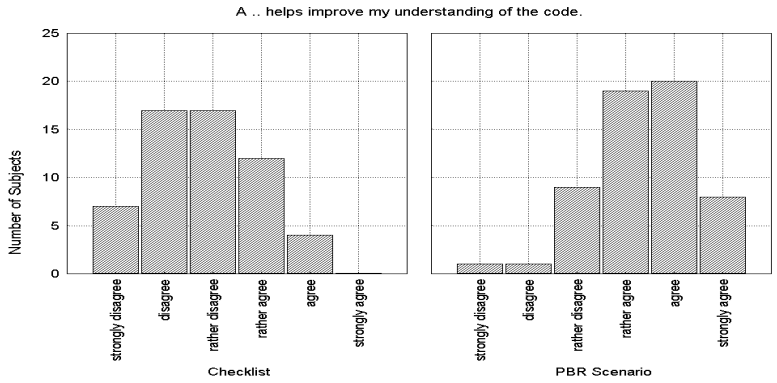

Figure 7: Histogram of Subject's Understanding of the Inspected Code Modules.

The above results tell a consistent story. Using PBR scenarios requires individual subjects to spend more effort for defect detection. Although this results in a higher checking rate, the cost per defect ratio of subjects using PBR for defect detection is acutally lower than for subjects using CBR. But the higher preparation effort together with the procedural support lead to an increased understanding of the inspected code module. This is then expected to lead to inspectors who can easily explain the defect that they found to their counterpart on the inspection team. Furthermore, it will take less effort to resolve false positives due to this enhanced understanding of the document. These would lead to less cost per defect for PBR teams when compared with CBR teams, which is the result that we obtained above.

We therefore can reject hypothesis H_{03}.

4.4 Cost per Defect for the Overall Inspection

We now consider the cost per defect results for the whole of the inspection. Figure 8 shows boxplots of the overall cost per defect.

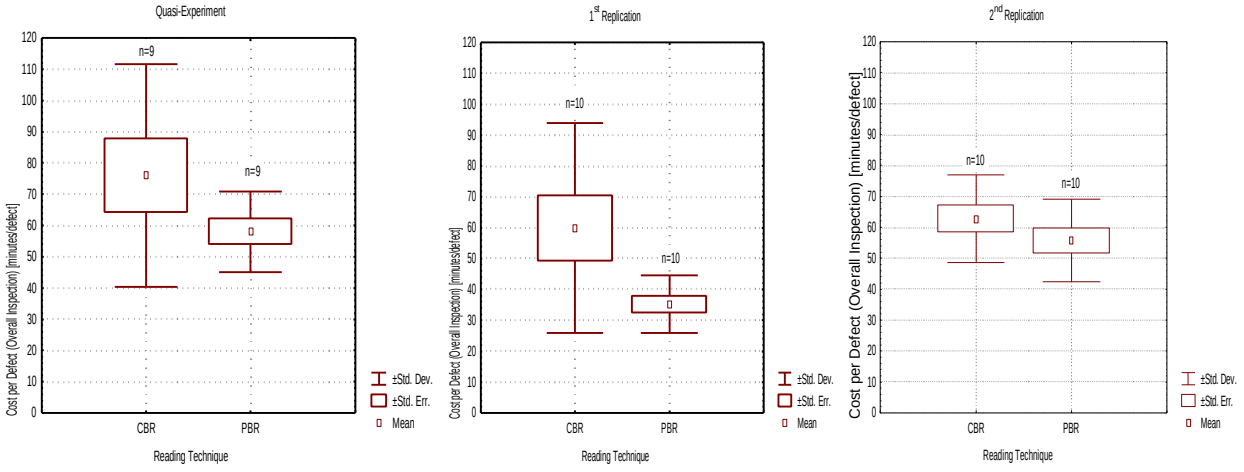

Figure 8: Box-Plots of the Overall Cost per Defect.

When looking at the overall cost per defect, the boxplots are consistent with the ones presented previously. PBR seems to have a lower cost per defect ratio and less variability than CBR. This result is consistent across all three studies. Hence, the extra effort that individual subjects spent using a PBR scenario is justified because in this case the inspection team reports more defects. We also observed less variability in the costs per defect when a team used a scenario for defect detection than a checklist, which may be due to the reduction of the influence of individual characteristics.

A test for carry-over effects was performed, and the results did not indicate any. The test results are summarised in Appendix F.

Table 13 presents the results of the matched-pair t-test for the cost per defect for teams for the overall inspection process.

	t-value	df	p-value (one-sided)
Quasi-Experiment	-1.86	8	0.05
1st Replication	-2.35	9	0.02
2nd Replication	-1.32	9	0.11

Table 13: t-Test Results of the Cost per Defect for the Overall Inspection.

Taking $\alpha = 0.1$ we can reject H_{04} for the quasi-experiment and the 1st replication. We cannot reject H_{04} for the 2nd replication.

We performed a meta-analysis as described previously and calculated $\sum (Z_j - \overline{Z})^2 = 0.32$. This value with 2 degrees of freedom results in p=0.85. Calculating the combination of the p-values according to Fisher's procedure results in:

$$P = -2 \times \sum_{i=1}^{k} \ln p_i = 18.08$$

Based on the c^2-distribution, this value of P results in a p-value of p=0.00012, which is statistically significant at an $\alpha = 0.1$ level. We can, therefore, reject the hypothesis H_{04} when looking at all three studies. The pooled result of the quasi experiment and its replications revealed that a team using the PBR technique for defect detection had a significantly lower cost per defect ratio than the team using CBR.

Table 14 shows the effect sizes of the three studies. It can be seen that, in fact, the 2nd replication has the lowest effect size value. The low effect size for the 2nd replication explains why the result of the statistical test turned out to be not statistically significant.

	S pooled	Hedges g	d
Quasi-Experiment	26.83	-0.67	-1.01
1st Replication	24.88	-0.97	-0.97
2nd Replication	13.81	-0.51	-0.54

Table 14: Effect Sizes of the Cost per Defect Ratio for the Overall Inspection.

For comparing and combining the effect sizes, we first checked the effect size homogeneity by calculating *Q*. The calculated value of *Q* is *Q* = 0.64 which leads to a p-value of p=0.73. Hence, we cannot reject the hypothesis that the effect sizes are homogeneous. This result allows us to combine the effect sizes. The combination of the effect size gives a value of 0.84. Considering our effect size threshold of 0.5, we can conclude that we have, in fact, found an effect of practical significance.

Given that the results for each of the individual phases, defect detection and meeting, point in the same direction, it is not surprising that cost per defect for PBR is lower than CBR for the whole of the inspection process. We can therefore reject hypothesis H_{04}.

4.5 Post-Hoc Analysis

We performed a post-hoc analysis to evaluate subjects' perceptions of ease of use of CBR and PBR. The rationale was to determine whether subjects are likely to revert back to using PBR if they apply CBR after PBR. If they find that CBR is easier to use, then this supports the argument that there is a reasonable amount of process conformance in the PBR→CBR ordering.

In a debriefing questionnaire, which subjects completed after each of the original quasi-experiment and the two replications, we asked them the following question: Which technique is the easiest one to use for defect detection?

The three response categories were: PBR, CBR, and their everyday-practice reading technique. We pooled the answers of all three studies. We found that 57% (34/60) selected CBR, only 18% (11/60) selected PBR, and 22% (13/60) selected their everyday practice reading technique. This provides some assurance of process conformance as described in Section 3.

However, even if there was contamination in the form of subjects who are supposed to be using CBR actually using PBR in the PBR→CBR ordering, that would be expected to improve the results of the CBR subjects. We found that PBR is better than CBR on all of our dependent variables. Therefore, if such a contamination existed, it was not sufficient enough to affect our results, and in fact, we could then consider that our results underestimate the beneficial impact of PBR when compared to CBR.

4.6 Sample Size Requirements for Future Studies

The planning of future studies that compare CBR with PBR can benefit from the estimates of effect size that we obtained. For repeated measures designs, we used the obtained mean effect sizes and the average correlation coefficients to estimate the minimal number of teams that would be necessary to attain a statistical power of 80% for one tailed tests at an alpha level of 0.1 using the paired t-test. These are summarised in Table 15.[25]

	Team Defect Detection Effectiveness	Cost per Defect (Defect Detection)	Cost per Defect (Meeting)	Cost per Defect (Overall Inspection)
Mean Effect Size	1.08	0.6	1.36	0.84
Estimated Sample Size	16	22	8	13

Table 15: Estimation of Sample Size (Number of Teams).

5 Threats to Validity

It is the nature of any empirical study that assumptions are made that later on restrict the validity of the results. Here, we list all these assumptions that impose threats to internal and external validity.

5.1 Threats to Internal Validity

Experiments in general and quasi-experiments in particular suffer from the problem that some factors may affect the dependent variables without the experimenter's knowledge [8]. This is referred to as a threat to internal validity. Although the threats to internal validity must be minimised, it is often not possible to

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

exclude them completely. For this study, we identified a potential history effect that may represent a threat to internal validity [8][45] that was not addressed during the study.

An experimenter cannot enforce subjects to apply a reading technique all of the time[26]. Inspectors start their defect detection activity by either reading the specification or the code documents and may already find defects during their first reading. Hence, it is plausible that a proportion of the reported defects are not directly attributable to the application of a particular reading technique, even if the subject using it applies it fully. However, as pointed out in [60], there is little possibility in quantifying this proportion.

5.2 Threats to External Validity

Threats to external validity limit the ability to generalise the results from an experimental study to a wider population under different conditions. There are three threats to external validity that we have identified for the current study:

- Single Organisation
  Our study was performed with subjects and code documents from a single organisation. While this enjoys greater external validity than doing studies with students in a "laboratory" setting, it is uncertain the extent to which the results can be generalised to other organisations.
- Inspection Process
  In this study, we assume that defect detection is an individual rather than a group activity. However, other inspection processes in industry may exist that consider defect detection a group activity, such as the one presented in [24].
- Type of Inspected Documents
  The code documents used in this study can be claimed to be representative of industrial code documents. However, we cannot generalise our findings for other type of documents, such as design or requirements documents.

To attain such generalisations, it is necessary to replicate the current study under different conditions.

6 Conclusions

Software inspection is considered as one of the most effective methods for software quality improvement. To exploit their full potential, a software inspection must call for a close and strict examination of the inspected artefact. This requires reading techniques that tell inspection participants what to look for and how to scrutinise a software artefact in a systematic manner. Apart from the fact that only few reading techniques are available, most recent work in this area focused on requirements documents because quality improvement of those documents promise a high return on investment. However, in industrial practice, the inspection of code documents still predominates, making the improvement of reading techniques for code documents an issue of contemporary concern.

In this paper, we have elaborated upon the perspective-based reading approach for code artefacts and compared its effectiveness and its cost per defect ratio to checklist-based reading. This was performed through a quasi-experiment and two close replications with professional developers of Bosch Telecom GmbH. During the three runs the subjects used the CBR approach as well as the PBR approach for defect detection in code modules of Bosch Telecom GmbH.

Our results indicate that the effectiveness of two-person teams using PBR is greater than when using CBR. Furthermore, we found that the cost per defect ratio using PBR is smaller than CBR during the

[26] Recall that we checked that subjects followed PBR by checking that they had produced the necessary abstractions and test cases. Also, we showed that a relatively large number of subjects found CBR to be easier to use than their everyday practice reading technique and also easier to use than PBR. Therefore, we do not expect that subjects when they are supposed to use CBR will revert to either of the other two, apparently more difficult, reading techniques. Furthermore, no performance feedback was provided after the initial treatment, and this should dampen the potential for continuing to use the first reading technique. Nevertheless, despite the above, we cannot be sure that the subjects followed the specified techniques *all of the time* during their defect detection activity and all detected defects are found because of applying a reading technique.

defect detection phase of inspections. Applying a PBR scenario helps improve a subject's perceived understanding of a code artefact, although it also consumes more effort on an inspector's behalf. This increased understanding leads to less 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 with CBR.

We provided the scenarios that we have used during the quasi-experiment and its replications in the appendix to this paper. Managers or technical staff wishing to pilot PBR within their organisations can take the perspectives and scenarios presented in this paper as a starting point for tailoring the PBR technique to their specific environment. Feedback from the inspectors would help him or her to improve the specificity and the detail of the guidelines. He or she may also consider incorporating other perspectives as we discussed in the paper.

Quasi-experimentation is an approach that can be applied when it is not feasible to assign subjects to treatments randomly. This situation can often be found in industrial settings. We are aware that quasi-experiments by their nature have difficulties in ruling out internal threats to validity. Some threats have their origin in the fact that this was not an experiment with students but in the field, that is, with software professionals from industry. We were careful to describe and address all of them in detail so that other researchers benefit from the lessons we have learned. They can, therefore, try to avoid these threats while replicating this quasi-experiment or setting up other empirical studies in this area. In general, we found quasi-experiments a good empirical approach that provides benefits for both researchers and practitioners. Other researchers may consider this kind of empirical study as one way of studying effects with real programmers in real industrial settings.

Our quasi-experiment adds to the current knowledge that more advanced reading techniques, such as perspective-based reading, optimise and improve software inspections by leveraging their effectiveness and reducing their cost. However, we need to collect more data to establish greater external validity to these results. We therefore encourage the external replication of this study in different environments by different researchers. A replication can take many forms, such as controlled experiments or case studies in industrial projects.

However, replication in general raises the question of how to compare and combine the results of the original study and the replications. We found meta-analysis techniques a useful tool for this purpose. Other researchers may consider these techniques in their arsenal of analysis approaches. This requires that researchers performing empirical research not only present results from statistical significant tests in their articles, e.g., p-values, but also compute and include the effect size and the number of subjects in their reporting.

7 Acknowledgements

We want to extend our very warm thanks to both the managers and developers at Bosch Telecom GmbH for their dedicated efforts in making possible as well as in participating in this study. We also wish to thank the anonymous reviewers for their valuable comments that have contributed to improving the content and presentation of this paper.

8 References

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

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

[3] V. Basili. Evolving and Packaging Reading Technologies. *Journal of Systems and Software*, 38(1), July 1997.

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

[5] L. Briand, K. El Emam, T. Fussbroich, and O. Laitenberger. Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects. *Proceedings of the Twentieth International Conference on Software Engineering*, pages 340–349. IEEE Computer Society Press, 1998.
[6] A. Brooks, J. Daly, J. Miller, M. Roper, and M. Wood. Replication of experimental results in software engineering. International Software Engineering Research Network (ISERN) Technical Report ISERN-96-10, University of Strathclyde, 1996.
[7] R. Brooks. Studying Programmer Behavior Experimentally: The Problems of Proper Methodology. *Communications of the ACM*, 23(4):207–213, April 1980.
[8] D. Campbell and J. Stanley. *Experimental and Quasi-Experimental Designs for Research*. Houghton Mifflin, Boston, 1966. ISBN 0-395-30787-2.
[9] B. Cheng and R. Jeffery. Comparing Inspection Strategies for Software Requirements Specifications. *Proceedings of the 1996 Australian Software Engineering Conference*, pages 203–211, 1996.
[10] Y. Chernak. A Statistical Approach to the Inspection Checklist Formal Synthesis and Improvement. *IEEE Transactions on Software Engineering*, 22(12):866–874, December 1996.
[11] D. A. Christenson, H. Steel, and A. Lamperez. Statistical Quality Control applied to Code Inspections. *IEEE Journal on Selected Areas in Communication*, 8(2):196–200, February 1990.
[12] W. Cochran. *Planning and Analysis of Observational Studies*. John Wiley & Sons, 1983.
[13] J. Cohen. *Statistical Power Analysis for the Behavioural Sciences*. Lawrence Erlbaum Associate Publishers, second edition, 1988.
[14] J. Cohen. A New Goal: Preventing Disease, Not Infection. *Science*, 262:1820-1821, 1993.
[15] J. Cohen. The Earth Is Round (p<.05). *American Psychologist*, 49:997–1003, 1994.
[16] T. Cook and D. Campbell. *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Rand McNally College Publishing Company, Chicago, 1979.
[17] M. Cowles and C. Davis. On the Origins of the .05 Level of Statistical Significance. *American Psychologist*, 37(5):553-558, 1982.
[18] B. Curtis. Measurement and Experimentation in Software Engineering. *Proceedings of the IEEE*, 68(9):1144–1157, September 1980.
[19] B. Curtis. By the Way, Did Anyone Study any Real Programmers? *Empirical Studies of Programmers: First Workshop*, pages 256–262. Ablex Publishing Corporation, 1986.
[20] J. Daly. *Replication and a Multi-Method Approach to Software Engineering Research*. PhD thesis, University of Strathclyde, 1996.
[21] H. Deitel and P. Deitel. *C How to program, 2nd ed.*. Prentice Hall, 1994.
[22] E. Doolan. Experience with Fagan's Inspection Method. *Software - Practice and Experience*, 22(2):173-182, 1992.
[23] M. Dyer. *The Cleanroom Approach to Quality Software Development*. John Wiley and Sons, Inc., 1992.
[24] M. Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3):182–211, 1976.
[25] M. Fagan. Advances in Software Inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751, July 1986.
[26] R. Fisher. Combining Independent Tests of Significance. *American Statistician*, 2(5), 1948.
[27] P. Fowler. In-process Inspections of Workproducts at AT&T. *AT&T Technical Journal*, 65(2):102–112, March 1986.
[28] P. Fusaro and F. Lanubile. A Replicated Experiment to Assess Requirements Inspection Techniques. *Empirical Software Engineering*, 2(1):39–57, 1997.
[29] J. Gibbons and J. Pratt. P-values: Interpretation and Methodology. *The American Statistician*, 29(1):20-25, 1975.
[30] T. Gilb and D. Graham. *Software Inspection*. Addison-Wesley Publishing Company, 1993.
[31] G. Glass, B. McGaw, M. L. Smith, Meta-Analysis in Social Research, Sage Publications, 1981.
[32] M. Graden, P. Horsley, and T. Pingel. The Effects of Software Inspections on a Major Telecommunications Project. *AT&T Technical Journal*, 65(3):32–40, May/June 1986.
[33] A. Greenwald. Within-Subjects Designs: To Use or Not to Use? *Psychological Bulletin*, 83(2), September 1976.
[34] J. Grizzle. The Two-period Chance-over Design and its Use in Clinical Trials. *Biometrics*, 21:314–320, 1965.

[35] W. Hays. *Statistics*. Hartcourt Brace, 1994.
[36] W. Hayes. Research Synthesis in Software Engineering: A Case for Meta-Analysis. To appear in *Proceedings of the International Symposium on Software Metrics*, 1999.
[37] L. Hedges and I. Olkin. *Statistical Methods for Meta-Analysis*. Academic Press, 1985.
[38] D. T. Heinsman and W. R. Shadish, Assignment Methods in Experimentation: When Do Nonrandomized Experiments Approximate Answers From Randomized Experiments?, *Psychological Methods*, 1(2):154-169, 1996.
[39] R. Henkel. *Tests of Significance*. Sage Publications, 1976.
[40] M. Hills and P. Armitage. The Two-period Cross-over Clinical Trial. *British Journal of Clinical Pharmacology*, 8:7– 20, 1979.
[41] M. Hwang. The Use of Meta-Analysis in MIS Research: Promises and Problems. *The DATA BASE for Advances in Information Systems*, 27(3):35-48, 1996.
[42] J. Cohen. *Applied Multiple Regression/Correlation Analysis for the Behavioural Sciences*. Lawrence Erlbaum Associates, Inc., Publishers, 1983.
[43] P. Jalote and M. Haragopal. Overcoming the NAH Syndrome for Inspection Deployment. In *Proceedings of the Twentieth International Conference on Software Engineering*, pages 371–378. IEEE Computer Society Press, 1998.
[44] P. Johnson and D. Tjahjono. Does Every Inspection Really Need a Meeting ? *Empirical Software Engineering*, 3:9-35, 1998.
[45] C. Judd, E. Smith, and L. Kidder. *Research Methods in Social Relations*. Holt, Rinehart and Winston, 6th edition, 1991.
[46] G. Keren. *A Handbook for Data Analysis in the Behavioural Sciences - Methodological Issues*, Chapter 19: Between- or Within-Subjects Design: A Methodological Dilemma. Lawrence Erlbaum Associates, 1993.
[47] B. Kernighan and D. Ritchie. *Programming in C*. Hanser Verlag, 1990.
[48] B. Kitchenham, S. Linkman, and D. Law. Critical Review of Quantitative Assessment. *Software Engineering Journal*, pages 43–53, March 1994.
[49] H. Kraemer and S. Thiemann. *How many Subjects*. Sage Publications, 1987.
[50] S. Kramer and R. Rosenthal: “Effect Sizes and Significance Levels in Small-Sample Research”. In R. Hoyle (ed.): *Statistical Strategies for Small Sample Research*, Sage Publications, 1999.
[51] S. Kusumoto, A. Chimura, T. Kikuno, K. Ichi Matsumoto, and Y. Mohri. A Promising Approach to Two-Person Software Review in an Educational Environment. *Journal of Systems and Software*, (40):115–123, 1998.
[52] O. Laitenberger and C. Atkinson. Generalizing Perspective-based Inspection to handle Object-Oriented Development Artefacts, Proceedings of the 21st International Conference on Software Engineering, Los Angeles, USA, 1999.
[53] O. Laitenberger and J.-M. DeBaud. An Encompassing Life-cycle Centric Survey of Software Inspection. Journal of Systems and Software (2000), also published as International Software Engineering Research Network (ISERN) Technical Report ISERN-98-14, Fraunhofer Institute for Experimental Software Engineering, http://www.iese.fhg.de/ISERN/pub/isern_biblio_tech.html. 1998.
[54] O. Laitenberger and J.-M. DeBaud. Perspective-based Reading of Code Documents at Robert Bosch GmbH. *Information and Software Technology*, 39:781–791, March 1997.
[55] L. Land, C. Sauer, and R. Jeffery. 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, 1997.
[56] R. Linger, H. Mills, and B. Witt. *Structured Programming: Theory and Practice*. Addison-Wesley Publishing Company, 1979.
[57] M. Lipsey. *Design Sensitivity*. Sage Publications, 1990.
[58] T. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.
[59] J. McCall. Quality Factors. In J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 2, pages 958– 969. John Wiley and Sons, 1994.
[60] J. Miller, M. Wood, and M. Roper. Further Experiences with Scenarios and Checklists. *Empirical Software Engineering*, 3(1):37–64, 1998.

[61] J. Miller. Applying Meta-Analytical Procedures to Software Engineering Experiments. To appear in *Journal of Systems and Software.*
[62] G. Myers. A Controlled Experiment in Program Testing and Code Walkthroughs / Inspections. *Communications of the ACM*, 21(9):760–768, September 1978.
[63] National Aeronautics and Space Administration. Software Formal Inspection Guidebook. Technical Report NASA-GB-A302, National Aeronautics and Space Administration, August 1993. http://satc.gsfc.nasa.gov/fi/fipage.html.
[64] Panel on Statistical Methods in Software Engineering. http://www.nap.edu/readingroom/books/statsoft/, 1993.
[65] D. Parnas and D. Weiss. Active Design Reviews: Principles and Practice. *Journal of Systems and Software*, 7:259–265, 1987.
[66] A. Porter, H. Siy, C. Toman, and L. Votta. An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development. *IEEE Transactions on Software Engineering*, 23(6):329–346, June 1997.
[67] A. Porter, L. Votta, and V. Basili. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. *IEEE Transactions on Software Engineering*, 21(6):563–575, June 1995.
[68] A. Porter and L. Votta. Comparing Detection Methods for Software Requirements Inspections: A Replication Using Professional Subjects. *Empirical Software Engineering*, 3:355-379, 1998.
[69] R. Lindsay and A. Ehrenberg. The Design of Replicated Studies. *The American Statistician*, 47(3):217–228, 1993.
[70] B. Regnell, P. Runeson, and T. Thelin. Are the Perspectives Really Different? Further Experimentation on Scenario-Based Reading of Requirements. Technical Report LUTEDEX(TETS-7172)/1-40/1999, Dept. of Communication Systems, Lund University, 1999.
[71] S. Rifkin and L. Deimel. Applying Program Comprehension Techniques to Improve Inspection, *Proceedings of the 19th Annual NASA Software Engineering Workshop*, NASA, 1994.
[72] R. Rosenthal. *Meta-Analytic Procedures For Social Research*. Sage Publications, 1984.
[73] R. Rosenthal and R. Rosnow. *Essentials of Behavioural Research: Methods and Data Analysis*. McGraw Hill Series in Psychology, 1991.
[74] R. Rosnow and R. Rosenthal. *Beginning Behavioural Research: A Conceptual Primer*. Prentice Hall International Editions, 1996.
[75] K. Sandahl, O. Blomkvist, J. Karlsson, C. Krysander, M. Lindvall, and N. Ohlsson. An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections. *Empirical Software Engineering*, 3:327-254, 1998.
[76] F. Schmidt. What Do Data Really Mean? Research Findings, Meta-Analysis, and Cumulative Knowledge in Psychology. *American Psychologist*, 47:1173–1181, 1992.
[77] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 1997.
[78] S. Siegel and J. Castellan. *Nonparametric Statistics For The Behavioural Sciences*. McGraw Hill, Inc., 2nd edition, 1988.
[79] E. Simpson. The Interpretation of Interaction in Contingency Tables. *Journal of the Royal Statistical Society*, B13:238-241, 1951.
[80] P. Spector. *Research Designs*. Number 07-023 in Quantitative Applications in the Social Sciences. Sage Publications, 1995.
[81] S. Shapiro and M. Wilk. A Comparative Study of Various Tests of Normality. *Journal of the American Statistical Association*, 63:1343–1372, 1968.
[82] M. Slatker, Y. B. Wu, N. S. Suzuki-Slatker, *, **, and ***; Statistical Nonsense At the .00000 Level, *Nursing Reasearch*, 40(4):248-249, 1991.
[83] V. Basili, F. Shull, and F. Lanubile. Using Experiments to Build a Body of Knowledge. Technical Report, University of Maryland, CS-TR-3983, 1998.
[84] T. van Dijk and W. Kintsch. *Strategies of Discourse Comprehension*. Academic Press, Orlando, 1984.
[85] L. Votta. Does Every Inspection Need a Meeting? *ACM Software Engineering Notes*, 18(5):107–114, December 1993.
[86] L. Votta. Does the Modern Code Inspection Have Value? *Presentation at the NRC Seminar on Measuring Success: Empirical Studies of Software Engineering*, March 1999. Available at: http://www.cser.ca/seminar/ESSE/slides/ESSE_Votta.PDF

[87] B. Winer, D. Brown, and K. Michels. *Statistical Principles in Experimental Design*, 3rd edition. McGraw Hill Series in Psychology, 1991.
[88] F. Wolf. *Meta-Analysis: Quantitative Methods for Research Synthesis*. SAGE University Paper, 1986.
[89] E. Youngs. Human Errors in Programming. *International Journal of Man-Machine Studies*, 6:361–376, 1974.
[90] G. Yule. Notes on the Theory of Association of Attributes in Statistics. *Biometrika*, 2:121-134, 1903.

9 Appendix A: The PBR Defect Detection Mechanisms

In this appendix we describe the assumptions behind PBR as precisely as possible. These assumptions are embedded in many scenario-based reading techniques, but have not always been clearly articulated. It is useful to spell out the embedded assumptions and assumed mechanisms that would make, in our case, PBR better than CBR, for the following reasons:

1. By making them explicit, it is possible to empirically evaluate the assumed mechanisms and assumptions to ensure that they really are the ones that are operating and that are causing the observed effects.
2. This will then allow us to gain a better understanding of the mechanisms that would make a reading technique work better than another. Such an improved understanding of the mechanisms can pave the road for improving the reading techniques themselves.

9.1 The Stated Underlying Defect Detection Assumptions in PBR

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 have a high probability of being detected, while the remainder of the defects have a relatively low probability of being detected by that perspective. 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 effective than individual CBR inspectors. We rather expect the effectiveness benefits of PBR to become apparent on the team level.

The posited mechanism in the literature that explains why PBR teams are likely to attain higher inspection effectiveness compared with CBR teams is depicted in Figure 9. Here, it is stated that because PBR focuses the inspectors' attention on different subsets of defects in a document, this will lead to a reduction in overlap (i.e., the same defects found by multiple inspectors). This is path (a) and represents a negative association between the extent to which the reading technique focuses the inspectors' attention on different defect subsets (Subset Focus) and the extent of overlapping defects (Defect Overlap). It is also stated that the reduction in defect overlap leads to a greater inspection effectiveness, which is shown by path (b) and is represented by a negative association. First we demonstrate that these are the assumptions made in the literature.

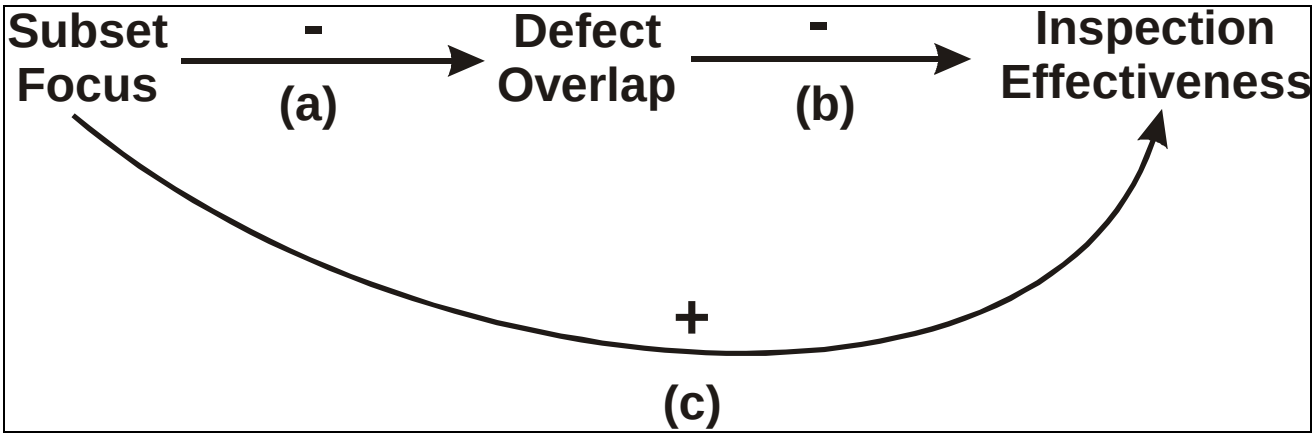

Figure 9: Posited mechanisms explaining the effect of PBR on inspection effectiveness.

Perhaps a fitting description of the assumption represented by path (a) in Figure 9 is given as follows "One main purpose of PBR is that the perspectives detect different kinds of defects in order to minimise the overlap among reviewers. […] If they detect the same defects, the overlap is not minimised and PBR does not work as it was meant to. […] The optimal solution is to use perspectives with no overlap" [70]. Figure 10 illustrates the effects expected on a team level for a two-person inspection team, which is also the context of our study.

Figure 10: Effect of different reading techniques at a team level.

The defect overlap reduction has been posited to increase the effectiveness of PBR when compared with CBR (path (b) in Figure 9). For example, in the context of scenario-based reading techniques, it has been stated that "Our hypothesis is that [a] systematic technique, specific and distinct responsibility inspections [such as PBR] achieve broader coverage and minimize reviewer overlap, resulting in higher fault detection rates and greater cost benefits than nonsystematic methods." [67][68],[27] and "The effort spent by [..] inspectors for defect detection can only be justified if they find different defects, i.e., if there is a small overlap between inspectors' perspectives." [54]

In the following subsection we show that the causal mechanism represented by paths (a) and (b) does not explain the effectiveness of PBR over CBR, and in fact only path (c) in Figure 9 explains the mechanism. This means that defect overlap has nothing to do with the impact of the use of PBR on inspection effectiveness, nor does it indicate how well defects are targeted by the perspectives.

9.2 Evaluating the Underlying Defect Detection Assumptions of PBR

In this subsection we elaborate a probabilistic model[28] for PBR and CBR with two inspectors. The purpose of the exposition is to determine whether the defect overlap reduction mechanism posited above would lead to increased defect detection effectiveness for PBR when compared with CBR.

9.2.1 Definitions

One of the main criteria for evaluating a software inspection is its effectiveness [5], which is defined as the proportion of actual defects found. To give this a frequentist interpretation, assume that a document has K defects and that the effectiveness of the inspection is 0.7 (i.e., 70% of the defects will be found). If one were to sample randomly with replacement from the K defects, then in the long run 70% of the sampled defects will be ones that are detected during the inspection. Therefore, in general when we say that a defect has a 0.7 probability of being detected, we mean that in the long run it will be selected 70% of the time. This interpretation has the convenient property that it is directly related to the common notion of inspection effectiveness.

When using a PBR perspective, a subset of the defects in the document have a high probability of being detected, while the remainder of the defects have a relatively low probability of being detected by that

[27] It should be noted that other authors, namely Johnson and Tjahjono [44], have also interpreted the Porter et al. article [67] to be explicitly stating that the reduction in overlap is a causal mechanism that explains the improved defect detection performance of a scanerio-based reading technique.

[28] In defining the model we make the standard assumptions that inspectors and defects are independent.

perspective. We can depict this in the two-perspective case (which is congruent with our study) as follows:

	Perspective A	Perspective B	CBR
$Subset_A$	$P_{A\text{-}HIGH}$	$P_{B\text{-}LOW}$	P
$Subset_B$	$P_{A\text{-}LOW}$	$P_{B\text{-}HIGH}$	

$Subset_A$ and $Subset_B$ constitute all defects in the document. This is congruent with the premise of PBR in that the totality of perspectives should provide comprehensive coverage of the defects in the inspected document. This is an explicitly stated objective of scenario-based reading techniques in general. For instance, in his discussion of reading families, such as DBR and PBR, Basili [3] states "each technique within the family is [...] (4) focused, in that it provides a particular coverage of the document, and a combination of techniques in the family provides coverage of the entire document".

Therefore, when a subject is using perspective A, a defect in $Subset_A$ has a high probability ($P_{A\text{-}HIGH}$) of being detected. In addition, a defect in $Subset_B$ has a low probability ($P_{A\text{-}LOW}$) of being detected. Therefore, perspective A is really targeting defects in $Subset_A$. The same argument can be made for perspective B above. This is the basic premise of PBR.

We let P_P be the proportion of defects in a document that are in $Subset_A$. Therefore, $1\text{-}P_P$ is the proportion of defects in $Subset_B$.

In the case of CBR, we would expect uniformity in the detection probabilities since no specific defects are targeted. Therefore, we can say for CBR:

$$P_{A-HIGH} = P_{A-LOW} = P \qquad \textbf{Eqn. 1}$$

where P is the probability of finding a defect using CBR.

We can also assume the following for PBR:

$$P_{A-HIGH} = P_{B-HIGH} = P_{HIGH} \qquad \textbf{Eqn. 2}$$

$$P_{A-LOW} = P_{B-LOW} = P_{LOW} \qquad \textbf{Eqn. 3}$$

This states that the efficacy of both perspectives in targeting a subset of defects is the same. This is empirically supported in [67].

The definition of PBR demands that the following two constraints also hold:

$$P_{HIGH} > P \qquad \textbf{Eqn. 4}$$

$$P_{LOW} < P \qquad \textbf{Eqn. 5}$$

Eqn. 4 is necessary since the premise of PBR is that it will target the relevant subset of defects, $Subset_A$, with a higher probability than a technique that does not embody targeting defects, such as CBR. In fact, if Eqn. 4 was not true, then by definition PBR *cannot* be better than CBR in terms of effectiveness. Therefore, if we do not believe Eqn. 4 then it is futile to compare PBR with CBR. Eqn. 5 is necessary because it states that defects that are not in the subset will receive little attention from a particular perspective compared with a technique that does not embody targeting defects. If this were *not* the case then there is no need for the perspectives to target a specific subset since even if $P_{HIGH} = P_{LOW}$ the PBR team will systematically perform as good as or better than CBR. Therefore, the basis of PBR as described by the above two inequalities is to focus more on a subset of defects, and focus less on other defects that are not targeted when compared to CBR.

Now let us define Eqn. 4 and Eqn. 5 more precisely:

$$P_{HIGH} = P + \Delta_1 \qquad \textbf{Eqn. 6}$$

$$P_{LOW} = P - \Delta_2 \qquad \textbf{Eqn. 7}$$

where the deltas are values indicating how much better than CBR the particular PBR implementation is at targeting defects.[29]

9.2.2 Testing Path (a)

In this subsection we show that the Overlap Reduction effect does not follow from the Subset Focus assumption (path (a) in Figure 9). We can define the probability of finding a defect by both inspectors (an overlap defect) using PBR as:

$$P(\text{Overlap Defect Using PBR}) = P_{HIGH}P_{LOW}P_P + P_{HIGH}P_{LOW}(1-P_P) = P_{HIGH}P_{LOW} \qquad \textbf{Eqn. 8}$$

$$P(\text{Overlap Defect Using PBR}) = (P+\Delta_1)(P-\Delta_2) = P^2 - P\Delta_2 + P\Delta_1 - \Delta_1\Delta_2 \qquad \textbf{Eqn. 9}$$

If the Overlap Reduction effect follows from the Subset Focus assumption, then we would expect that as the PBR perspectives become better at focusing on a specific subset of defects, the overlap as defined in Eqn. 9 would decrease. Let us say that we have constructed improved PBR perspectives that are superior to traditional perspectives by means of an increased probability to detect defects in the target subset, and therefore the following holds:

$$P'_{HIGH} = P + \Delta_1 + T \qquad \textbf{Eqn. 10}$$

where the additional positive T indicates that the PBR definition is superior to the normal PBR as defined by Eqn. 6 at focusing on a target subset of defects. This also means that different perspectives are focusing better on different defects. If we subsitute Eqn. 10 into Eqn. 8, we get the following:

[29] It is assumed that all PBR implementations would have deltas that are positive.

$$(P+\Delta_1+T)(P-\Delta_2)=(P^2-P\Delta_2+P\Delta_1-\Delta_1\Delta_2)+T(P-\Delta_2)$$ **Eqn. 11**

This indicates that the improved PBR, in that it is better able to focus on a specific subset of defects, will actually result in an *increased* overlap rather than a reduced one. This example demonstrates that the first set of assumptions about the relationship between focus and overlap in PBR are not necessarily true. It also shows that defect overlap is not a good indicator of the extent to which different perspectives focus on different defects.

9.2.3 Testing Path (b)

The probability of finding a defect by both inspectors using CBR is given by:

$$P(\text{Overlap Defect Using CBR})=P^2$$ **Eqn. 12**

If we define the following constraint, which we shall return to later:

$$?_2 > ?_1 \times \frac{P}{P_{HIGH}}$$ **Eqn. 13**

then we can expand this inequality as follows:

$$?_2 > \frac{?_1 \times P}{\Delta_1 + P}$$ **Eqn. 14**

$$P\times\Delta_2 > (P\times\Delta_1)-(\Delta_1\times\Delta_2)$$ **Eqn. 15**

$$\Delta_2 > \Delta_1 - \frac{\Delta_1\times\Delta_2}{P}$$ **Eqn. 16**

$$\Delta_2 + P > \Delta_1 - \frac{\Delta_1\times\Delta_2}{P} + P$$ **Eqn. 17**

$$P > \left(\frac{\Delta_1}{P}+1\right)\times(P-\Delta_2)$$ **Eqn. 18**

$$\left(\frac{P}{P-\Delta_2}-1\right) > \frac{\Delta_1}{P}$$ **Eqn. 19**

$$\frac{\Delta_2}{P-\Delta_2} > \frac{\Delta_1}{P}$$ **Eqn. 20**

$$(\Delta_1 \times (P - \Delta_2)) - (P \times \Delta_2) < 0 \qquad \textbf{Eqn. 21}$$

$$(\Delta_1 \times (P - \Delta_2)) - (P \times \Delta_2) + P^2 < P^2 \qquad \textbf{Eqn. 22}$$

$$(\Delta_1 \times P) - (\Delta_1 \times \Delta_2) - (P \times \Delta_2) + P^2 < P^2 \qquad \textbf{Eqn. 23}$$

$$(P + \Delta_1) \times (P - \Delta_2) < P^2 \qquad \textbf{Eqn. 24}$$

$$P_{HIGH} \times P_{LOW} < P^2 \qquad \textbf{Eqn. 25}$$

Eqn. 25 tells us that the probability of finding a defect by both inspectors using PBR is less than for CBR. However, this is only under the condition specified in Eqn. 13. Eqn. 13 makes intuitive sense since it places a minimal threshold on Δ_2. The larger the value of Δ_2 the more focused a PBR perspective is. The smaller the value of Δ_2, then the greater the probability that a perspective will be finding defects that are targeted by the other perspective. Eqn. 13 specifies the constraint in terms of a minimal fraction of Δ_1.

Now we consider the overall probability of detecting a defect by the PBR team and the CBR team. We have:

$$P(\text{Detecting a Defect Using PBR}) = P_{HIGH} + P_{LOW} - (P_{HIGH} \times P_{LOW}) \qquad \textbf{Eqn. 26}$$

And the overall probability of detecting a defect by the CBR team is:

$$P(\text{Detecting a Defect Using CBR}) = (2 \times P) - P^2 \qquad \textbf{Eqn. 27}$$

If we define the following constraint, which we shall return to later:

$$\Delta_1 > \frac{\Delta_2 \times (1 - P)}{1 - P_{LOW}} \qquad \textbf{Eqn. 28}$$

then we can expand this inequality as follows:

$$\Delta_1 \times (1 - P_{LOW}) > \Delta_2 \times (1 - P) \qquad \textbf{Eqn. 29}$$

$$(\Delta_1 \times (1 - P + \Delta_2)) + (\Delta_2 \times (P - 1)) > 0 \qquad \textbf{Eqn. 30}$$

$$(2\times P)-P^2+(\Delta_1\times(1-P+\Delta_2))+(\Delta_2\times(P-1))>(2\times P)-P^2 \quad \textbf{Eqn. 31}$$

$$(2\times P)-P^2+\Delta_1-(\Delta_1\times P)+(\Delta_1\times\Delta_2)+(\Delta_2\times P)-\Delta_2>(2\times P)-P^2 \quad \textbf{Eqn. 32}$$

$$(P+\Delta_1)+(P-\Delta_2)-((P+\Delta_1)\times(P-\Delta_2))>(2\times P)-P^2 \quad \textbf{Eqn. 33}$$

$$P_{HIGH}+P_{LOW}-(P_{HIGH}\times P_{LOW})>(2\times P)-P^2 \quad \textbf{Eqn. 34}$$

Eqn. 34 tells that the probability of finding a defect using PBR is greater than that of CBR, but only under the condition stipulated in Eqn. 28. Eqn. 28 makes intuitive sense, in that it sets a lower limit on the probability of detecting the targeted defects using PBR.

Based on the defect overlap reduction mechanism in the literature (i.e., path (b) in Figure 9), we would expect that the condition specified in Eqn. 13, which is necessary for the defect overlap reduction effect of PBR, to be a precondition for Eqn. 28. However, it will be noted that the two constraints that we have identified, namely Eqn. 28 and Eqn. 13, are not the same. This should not be surprising, however, as they are addressing different issues. The reduction in overlap due to using PBR could also increase the number of unique defects that are discovered, but not necessarily so. In order to increase the number of discovered defects, it is not necessary to minimise the defect overlap at all. They are two separate issues that do not necessarily follow each other.

We illustrate this more clearly below. We first define the following differences such that if the value is positive then that indicates that PBR is performing better than CBR, and negative otherwise:

Difference(Defect Detection) = P(Detecting a Defect Using PBR) – P(Detecting a Defect Using CBR) **Eqn. 35**

Difference(Overlap) = P(Overlap Defect Using CBR) – P(Overlap Defect Using PBR) **Eqn. 36**

Based on the posited defect overlap reduction mechanism in Figure 9, one would expect that whenever Eqn. 36 is positive, then it follows that Eqn. 35 is also positive for any implementation of PBR.

In the graph in Figure 11 we plot the differences above on the x and y axes. We take P=0.5 for the plots. The value of P_{HIGH} is varied, and is represented as different lines. We use values of 0.6, 0.7, 0.8, and 0.9 (all have to be larger than P). We then generated values for P_{LOW} varying from 0 to 0.499. Each value of P_{LOW} is used to calculate the x and y values for the given value of P_{HIGH}, and these are the values that determine the co-ordinates on the graph. Such a plot allows us to represent all of the information in one plot.

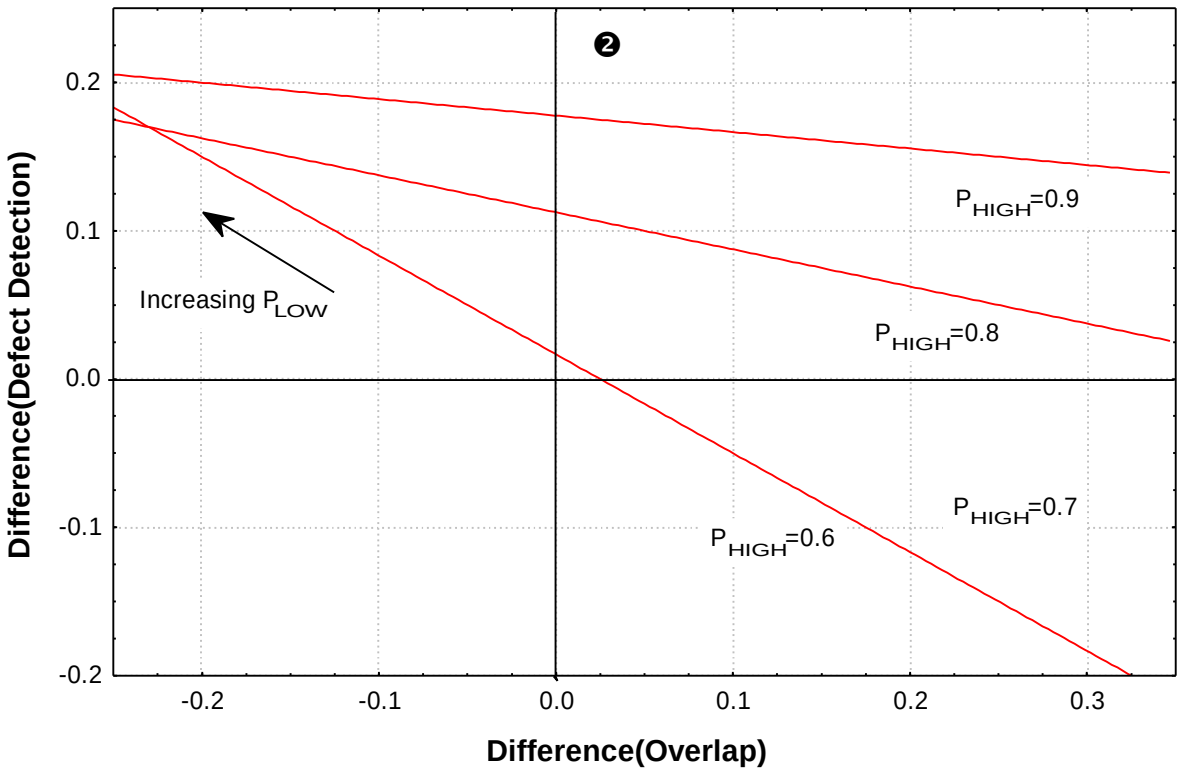

Figure 11: Difference in Defect Detection against Difference in Overlap.

In quadrant (1), inequality Eqn. 13 is not satisfied, but Eqn. 28 is satisfied. In such a case P_{LOW} is so high (i.e., Δ_2 is quite low) and therefore the overlap from using PBR is larger than that for using CBR. But because Δ_2 is quite low, the inequality in Eqn. 28 is easily satisfied, and therefore PBR finds more defects than CBR.

In quadrant (3), inequality Eqn. 13 is satisfied, but the inequality in Eqn. 28 is not satisfied. In such a case P_{LOW} is so low (i.e., Δ_2 is quite large) and therefore there is very little overlap when using PBR. But, because Δ_2 is high, inequality Eqn. 28 is not satisfied, and therefore the overall performance of PBR in terms of defect detection is weak.

In quadrant (2) both inequalities are satisfied. This means that PBR achieves both outcomes: a reduction in defect overlap and an increase in effectiveness. Of course, according to the literature, out of the three quadrants only this quadrant should have any values.

What the above shows is that it is fairly plausible to attain one of the effects of PBR only, but not the other. This also means that the "reduction in defect overlap" hypothesis, while it may co-occur with an overall increase in defect detection, is certainly not a causal mechanism that can explain it.

To make the point more concretely, below we present the results of Monte Carlo simulations that illustrate the lack of causality[30]. The simulations are done for a document that has 30 defects with half in $Subset_A$ and the other half in $Subset_B$, whereby we also have two inspectors. In all cases, 1000 inspections were

[30] While the results of the simulation can be derived from the above equations, it was felt that concrete examples would further clarify the point.

simulated. We present the results in terms of box and whisker plots. The point is the mean. The box represents one standard deviation, and the whiskers the minimum and maximum values.

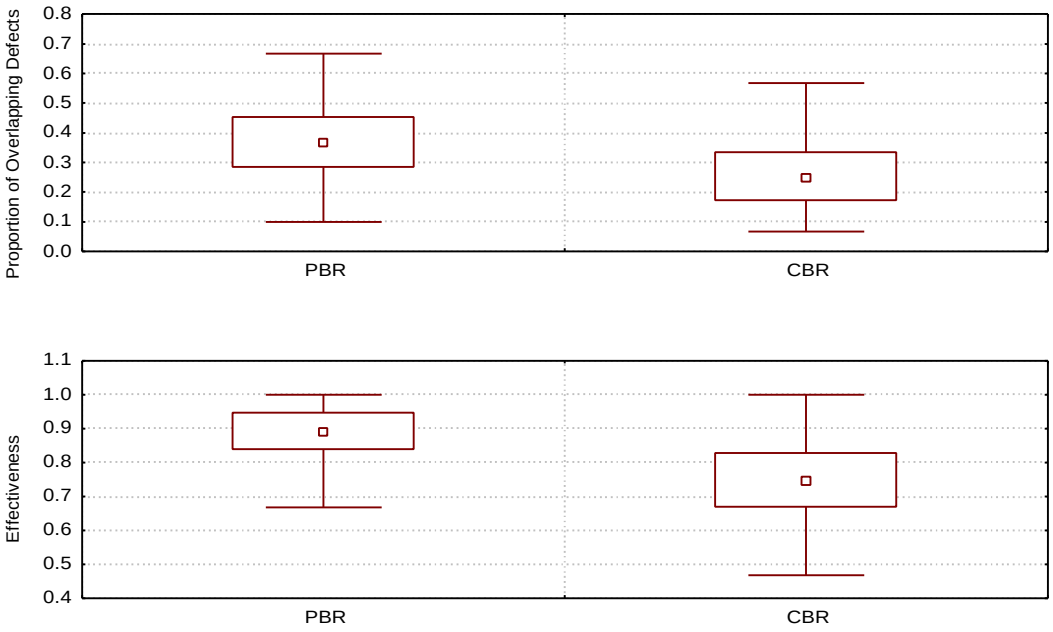

Figure 12: Results of a simulated inspection with P_{HIGH} = 0.8 and P_{LOW} = 0.46.

The plot in Figure 12 shows an inspection in quadrant (1). This one had a P_{HIGH} of 0.8 and a P_{LOW} of 0.46. As can be seen here, the mean proportion of overlapping defects is actually greater for PBR than for CBR, but its mean effectiveness is also larger. This is an example of PBR not reducing overlapping defects, but still being more effective.

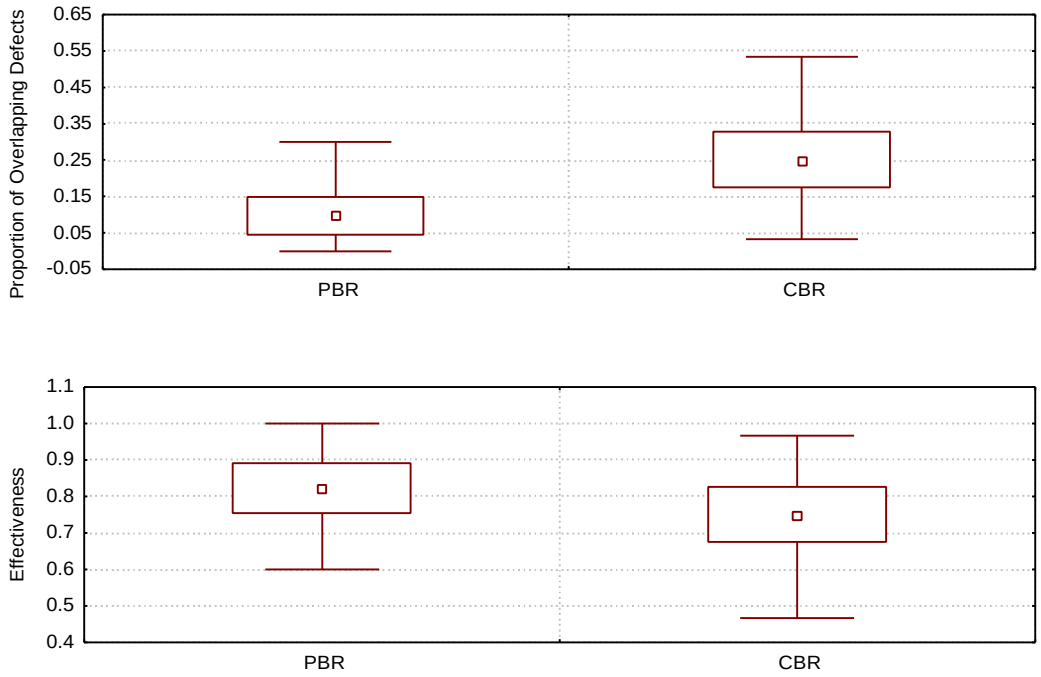

Figure 13: Results of a simulated inspection with P_{HIGH} =0.8 and P_{LOW} =0.12.

The plot in Figure 13 shows an inspection in quadrant (2). This one had a P_{HIGH} of 0.8 and a P_{LOW} of 0.12. In this particular case PBR had less overlap in defects (the top panel with PBR exhibiting a lower mean), and also had a larger effectiveness.

The plot in Figure 14 shows an inspection in quadrant (3). This one had a P_{HIGH} of 0.6 and a P_{LOW} of 0.1. For this particular inspection, PBR did have less overlapping defects than for CBR, but its effectiveness was also lower than that for CBR.

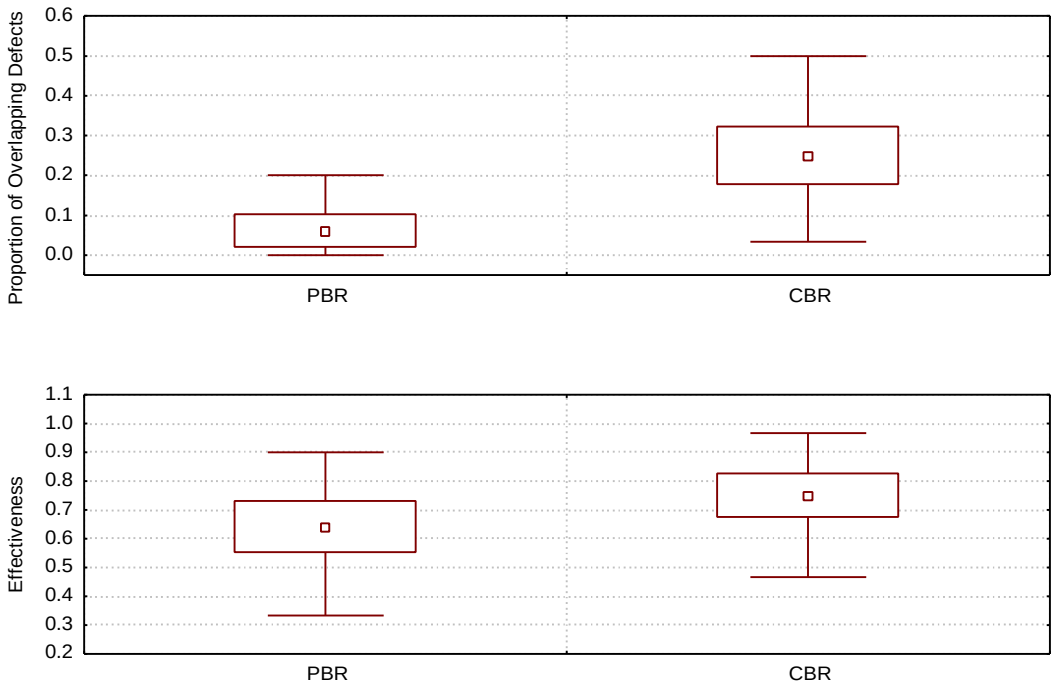

Figure 14: Results of a simulated inspection with P_{HIGH} = 0.6 and P_{LOW} =0.1.

9.3 Summary

The above illustrates that the postulated mechanism that explains the effectiveness benefits of PBR, as shown through path (a) and path (b) in Figure 9, is incorrect. In fact, the benefits of PBR are explained through path (c), where if the inequality in Eqn. 28 is satisfied, then the probability of finding a unique defect through PBR is greater than for CBR. This inequality sets a minimal value to Δ_1 in terms of a fraction of Δ_2.

10 Appendix B: The Code Analyst Scenario

Code Analyst Scenario

Assume you have the role of a code analyst. As a code analyst you have to ensure that the right functionality is implemented in the code.

In doing so, take the code document and determine the functions that are implemented in this code module. Determine the dependencies among these functions and document them in the form of a call graph.

Starting with the functions at the leaves of the call graph, determine the implemented operation of each function in the following manner.

1. Identify (sequences of) assignment operations and highlight them. Determine the meaning of these (sequences of) assignment operations.
2. Combine the (sequences of) assignment operations by taking into account conditions and loops. Determine the meaning of the larger structures.

Repeat 2 recursively until you have determined the operation that is implemented in a function. Document the operation of each function.

Hint: It is useful to describe the different levels of abstraction as formal as possible. Using implicit knowledge, such as, assumptions of global variable values, should be avoided.

Check for each function, whether your description matches the description that is given in code comments and the description in the specification. If differences exist, check whether there is a defect. Document each defect you detect on the defect report form.

While following the instructions ask yourself the following questions:

1. Does the operation described in the code match the one described in the specification?
2. Are there operations described in the specification that are not implemented?
3. Is data (i.e., constants and variables) used in a correct manner?
4. Are all the calculations performed in a correct manner?
5. Are interfaces between functions used correctly?

11 Appendix C: The Tester Scenario

Tester Scenario

Assume you have the role of a tester. As a tester you have to ensure that the functionality implemented in the code is correct.

In doing so, take the code document and determine the functions that are implemented in this code module. Determine the dependencies among these functions and document them in the form of a call graph.

Starting with the functions at the leaves of the call graph, determine for each function a set of test cases that allow you to stimulate the operation of the function. The set of test cases should allow you to check each branch of the function as well as the loops. Document some of the test cases.

Assume you are executing the function with your test cases as input values (mental simulation). Verify, whether each function behaves according to its specification and the comments given in the code. If differences occur, check whether there is a defect or not. Document each defect you detect on the defect report form.

While following the instructions ask yourself the following questions:

1. Do you have the necessary information to identify a test case (e.g., are all constant values and interfaces defined)?
2. Are branch conditions used in a correct manner?
3. Can you generate test cases for each branch and each loop? Can you traverse all branches by using specific test cases?
4. Is allocation and deallocation of memory used correctly?

12 Appendix D: Testing for Carry-over Effects

Below we describe the approach for testing for carry-over effect. We have to state that a carry-over effect from treatment A to treatment B may have two components. One is due to unique practice effects of treatment A (which may be considered as a learning effect) and the other refers to the switch in conditions from treatment A to treatment B. The significance test for carry-over effects is legitimate regardless of whether either one or both of these effects may be present. However, the test is incapable of discriminating between them.

To perform the test for carry-over effects, we have restructured our data set as shown in the following table.

	(CBR ⟶ PBR)-group		
Team	Period 1 (CBR)	Period 2 (PBR)	SUM (CBR+ PBR)
1	1	3	4
2	4	3	7
3	6	4	10
…			
Mean	$\overline{Y}_{1.1}$	$\overline{Y}_{1.2}$	

	(PBR ⟶ CBR)-group		
Team	Period 1 (PBR)	Period 2 (CBR)	SUM (CBR+ PBR)
6	2	3	5
7	1	6	7
8	1	4	5
…			
	$\overline{Y}_{2.1}$	$\overline{Y}_{2.2}$	

Table 16: Structure of the Data Set.

To test for carry-over effects, one performs an analysis of variance comparing the variability between sequences (or orders) with that of subjects within sequences (or orders). Let $\overline{Y}_{i.k}$ be the mean of all scores for order *i* and period *k*. In our case, we have two orders (PBR ⟶ CBR, CBR ⟶ PBR) and two periods (Period 1 and Period 2). Let $\overline{Y}_{ij.}$ be the total of the two scores for subject *j* in sequence.

Grizzle [34] reports a convenient formula for $SS_{Sequences} = SS_{Carry\text{-}over}$:

$$SS_{Carryover} = \frac{n_1 \times n_2}{2 \times (n_1 + n_2)} \times \left(\overline{Y}_{1.1} + \overline{Y}_{1.2} - \overline{Y}_{2.1} - \overline{Y}_{2.2}\right)^2 \qquad \textbf{Eqn. 20}$$

The degrees of freedom equals the number of sequences minus 1, that is, $df_{Carry\text{-}over}$ = number of sequences -1 = 1. The Hills and Armitage error sum of squares [38] for testing for sequence effects can be rewritten as

$$SS_{\text{Subjects within sequences}} = 0.5 \times \left[\sum_j Y_{1j.}^{\ 2} + \sum_j Y_{2j.}^{\ 2} - \frac{\left(\sum_j Y_{1j.} \right)^2}{n_1} - \frac{\left(\sum_j Y_{2j.} \right)^2}{n_2} \right] \qquad \textbf{Eqn. 21}$$

Note that $df_{\text{Subjects within sequences}} = n_1+n_2-2$.

Based on these formulae, we can compute $MS_{\text{Carry-over}}$ and $MS_{\text{Subjects within Sequences}}$ in the following manner:

$$MS_{\text{Carry-over}} = SS_{\text{Carry-over}} \qquad \textbf{Eqn. 22}$$

$$MS_{\text{Subjects within Sequences}} = \frac{SS_{\text{Subjects within Sequences}}}{df_{\text{Subjects within Sequences}}} \qquad \textbf{Eqn. 23}$$

This allows us to compute the F-value as follows:

$$F(df_{\text{Sequence}}, df_{\text{Subjects within Sequences}}) = \frac{MS_{\text{Carryover}}}{MS_{\text{Subjects within Sequences}}} \qquad \textbf{Eqn. 24}$$

13 Appendix E: Data for the Meta-Analysis

	p-value (one sided)	Z-Value	ln p	Correlation Coefficient
Quasi-Experiment	0.0074	2.44	-4.9	0.77
1st Replication	0.10	1.27	-2.28	0.78
2nd Replication	0.02	2.05	-3.89	0.03[31]

Table 17: Data for Comparing and Combining p-values of the Team Defect Detection Effectiveness. The $\overline{Z}$ is 1.92. The chi-square value 0.7098.

	p-value (one sided)	Z-Value	ln p	Correlation Coefficient
Quasi-Experiment	0.11	1.22	-2.199	0.62
1st Replication	0.043	1.72	-3.15	-0.02[32]
2nd Replication	0.24	0.7	-1.44	0.03[33]

Table 18: Data for Comparing and Combining p-values of the Cost per Defect for the Defect Detection Phase. The $\overline{Z}$ is 1.213. The chi-square value 0.52.

	p-value (one sided)	Z-Value	ln p	Correlation Coefficient
Quasi-Experiment	0.0004	3.4	-7.797	0.55
1st Replication	0.035	1.81	-3.34	0.48
2nd Replication	0.016	2.16	-4.16	0.17

Table 19: Data for Comparing and Combining p-values of the Cost per Defect for the Meeting Phase. The $\overline{Z}$ is 2.46. The chi-square value 1.39.

[31] The results from one team had a large impact on this correlation coefficient. When removing this observation the correlation coefficient is 0.20. The associated effect size is 0.85, which is lower than the one presented previously. This derives from the fact that removing this data point also changes the mean values and the standard deviation. However, the removal of this observation does not alter our meta-analysis conclusions, and has negligible impact on the total effect size.

[32] The results from one team had a large impact on this correlation coefficient. When removing this observation the correlation coefficient is 0.47. The associated effect size is 0.93, which is larger than the one presented previously. The removal of this observation actually improves the results in favor of PBR, which means that the results we present in the body of the paper are conservative.

[33] The results from one team had a large impact on this correlation coefficient. When removing this observation the correlation coefficient is 0.26. The associated effect size is 0.06, which is quite smaller than the one presented previously. This can be explained by the fact that the removal of this observation actually results in similar mean values of PBR and CBR. Thus the effect size becomes smaller. Although the removal of this observation reduces the total effect size, the results are still in favour of PBR and our meta-analysis conclusions still hold.

	p-value (one sided)	Z-Value	ln p	Correlation Coefficient
Quasi-Experiment	0.05	1.65	-2.99	0.65
1st Replication	0.02	2.03	-3.83	0.21
2nd Replication	0.11	1.23	-2.21	0.27

Table 20: Data for Comparing and Combining p-values of the Cost per Defect for the Overall Inspection. The $\overline{Z}$ is 1.64. The chi-square value 0.3203.

14 Appendix F: Results of the Carry-over Effect Tests

In this appendix we present the results of the carry-over effect analysis for each of our four dependent variables.

14.1 Carry-over Effect for Defect Detection Effectiveness

The table below shows the analysis of variance results for carry-over effects for the dependent variable: defect detection effectiveness. This indicates that no carry-over effect is discernable (all F values are not significant at an alpha level of 0.1).

	$SS_{Carry\text{-}over}$	$SS_{Subjects\ within\ Sequences}$	$Df_{Subjects\ within\ Sequences}$	$MS_{Carry\text{-}over}$	$MS_{Subjects\ within\ Sequences}$	$F(df_{Sequence}, df_{Subjects\ within\ sequences})$	p-value
Quasi-Experiment	0.68	6.99	7	0.68	0.9985	0.68	0.56
1st Replication	0.38	9.10	8	0.38	1.14	0.33	0.42
2nd Replication	0.91	4.02	8	0.91	0.50	1.82	0.79

Table 21: Results of Testing for Carry-over Effect (Defect Detection Effectiveness).

14.2 Carry-over Effect of the Cost per Defect for the Defect Detection Phase

The table below shows the analysis of variance results for carry-over effects for the dependent variable: cost per defect for the defect detection phase. This indicates that no carry-over effect is discernable (all F values are not significant at an alpha level of 0.1).

	$SS_{Carry\text{-}over}$	$SS_{Subjects\ within\ Sequences}$	$Df_{Subjects\ within\ Sequences}$	$MS_{Carry\text{-}over}$	$MS_{Subjects\ within\ Sequences}$	$F(df_{Sequence}, df_{Subjects\ within\ sequences})$	p-value
Quasi-Experiment	94.4733	56605.27	7	94.47329	8086.466	0.01	0.56
1st Replication	2623.80	29454.92	8	2623.802	3681.865	0.71	0.40
2nd Replication	3141.29	22911.5	8	3141.291	2863.937	1.10	0.76

Table 22: Results of Testing for Carry-over Effect (Cost per Defect for the Defect Detection Phase).

14.3 Carry-over Effect of the Cost per Defect for the Meeting Phase

The table below shows the analysis of variance results for carry-over effects for the dependent variable: cost per defect for the Meeting phase. This indicates that no carry-over effect is discernable (all F values are not significant at an alpha level of 0.1).

	$SS_{Carry\text{-}over}$	$SS_{Subjects\ within\ Sequences}$	$Df_{Subjects\ within\ Sequences}$	$MS_{Carry\text{-}over}$	$MS_{Subjects\ within\ Sequences}$	$F(df_{Sequence}, df_{Subjects\ within\ sequences})$	p-value
Quasi-Experiment	1261.43	84089.46	7	1261.43	12012.78	0.11	0.25
1st Replication	4816.45	46673.47	8	4816.45	5834.18	0.83	0.61
2nd Replication	2557.84	31415.56	8	2557.84	3926.95	0.65	0.56

Table 23: Results of Testing for Carry-over Effect (Cost per Defect for the Meeting Phase).

14.4 Carry-over Effect of the Cost per Defect for the Overall Inspection

The table below shows the analysis of variance results for carry-over effects for the dependent variable: cost per defect for the overall inspection. This indicates that no carry-over effect is discernible (all F values are not significant at an alpha level of 0.1).

	$SS_{Carry\text{-}over}$	$SS_{Subjects\ within\ Sequences}$	$Df_{Subjects\ within\ Sequences}$	$MS_{Carry\text{-}over}$	$MS_{Subjects\ within\ Sequences}$	$F(df_{Sequence}, df_{Subjects\ within\ sequences})$	p-value
Quasi-Experiment	21.06	698.85	7	21.06	99.84	0.21	0.34
1^{st} Replication	25.65	560.52	8	25.65	70.06	0.37	0.44
2^{nd} Replication	5.34	169.39	8	5.34	21.17	0.25	0.37

Table 24: Results of Testing for Carry-over Effect (Cost per Defect for the Overall Inspection).

Oliver Laitenberger is 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.

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.

Thomas G. Harbich received his diploma degree in physics at the University of Regensburg, Germany in 1977 and his Ph.D. degree in theoretical solid state physics at the University of Stuttgart in 1982. He joined ANT-Nachrichtentechnik GmbH, Backnang Germany (now BOSCH Telecom GmbH) in 1983 where he held different engineering positions in the field of telecommunications systems. Currently he is leading a department for software development with the main focus on embedded software for transmission systems and access networks.