#. A Survey of Software Inspection Technologies

Oliver Laitenberger

Fraunhofer Institute for Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
Email: Oliver.Laitenberger@iese.fhg.de

Abstract

Software inspection is a proven method that enables the detection and removal of defects in software artifacts as soon as these artifacts are created. It usually involves activities in which a team of qualified personnel determines whether the created artifact is of sufficient quality. Detected quality deficiencies are subsequently corrected. In this way an inspection can not only contribute towards software quality improvement, but also lead to significant budget and time benefits. These advantages have already been demonstrated in many software development projects and organizations.
After Fagan's seminal paper presented in 1976, the body of work in software inspection has greatly increased and matured. This survey is to provide an overview of the large body of contributions in the form of incremental improvements and/or new methodologies that have been proposed to leverage and amplify the benefits of inspections within software development and even maintenance projects. To structure this large volume of work, it introduces, as a first step, the core concepts and relationships that together embody the field of software inspection. In a second step, the survey discusses the inspection-related work in the context of the presented taxonomy.
The survey is beneficial for researchers as well as practitioners. Researchers can use the presented survey taxonomy to evaluate existing work in this field and identify new research areas. Practitioners, on the other hand, get information on the reported benefits of inspections. Moreover, they find an explanation of the various methodological variations and get guidance on how to instantiate the various taxonomy dimensions for the purpose of tailoring and performing inspections in their software projects.

Keywords

Software Quality, Defect Costs, Software Inspection, Reading Techniques

Introduction

It has been more than 20 years since Michael Fagan described the inspection approach in the software domain [40][1]. Since then many others, such as Gilb and Graham [47], have fine-tuned the inspection method to make it an even more cost-effective instrument for tackling quality deficiencies and defect costs. In fact, it has been claimed that inspection technologies can lead to the detection and correction of anywhere between 50 percent and 90 percent of the defects [47]. Moreover, early defect detection and removal improve the predictability of software projects and help project managers stay within schedule, since problems are unveiled throughout the early development phases. Most of the stated benefits of inspections have already been demonstrated in many projects and organizations. However, most of the published inspection work has not been integrated into a broader context, that is, into a coherent body of knowledge, hence making the work difficult to reconcile and evaluate for software practitioners. To provide a systematic and encompassing view of the research and practice in software inspection, this survey portrays the current state of the art and practice as published in available software inspection publications.

The survey consists of two main parts. The first includes a taxonomy of the core concepts and relationships that together embody the software inspection field. This taxonomy is organized around four primary dimensions -- technical, economics, organizational, and tools -- with which we attempt to characterize the nature of software inspection. While these primary dimensions are most relevant for the major areas of software development, we elicited from the literature particular sub-dimensions that are principal for the work in the software inspection area. The second part describes the various contributions and integrates them into the taxonomy, taking into account their specific particularities. This can make it much easier for researchers and practitioners to get an overview of relevant inspection work including its empirical validation.

Researchers and practitioners can profit from this survey in different ways. Researchers can use the presented taxonomy to characterize new work in the inspection-related field and identify fruitful areas for future research. Practitioners, on the other hand, find information about inspection-related benefits. This information may pave the road for the use of the inspection method in projects. Moreover, they find a road-map in the form of a taxonomy that helps them focus quickly on the best suited inspection approach adapted to their particular environment.

---

[1] In this survey, we consider inspection to be an approach that involve a well-defined and disciplined process in which qualified personnel analyse a software product using a reading technique for the purpose of detecting defects. We acknowledge that others may define the term "software inspection" differently.

Integration of Software Inspection in the Development Context

One prevalent reason for the use of inspection technology in software projects is the inevitability of defects. Even with the best development technologies in place, defects cannot be completely avoided. This stems from the fact that software development is a human-based activity, and, thus, prone to defects. A defect can be characterized as any product anomaly, that is, any deviation from the required quality properties that needs to be tracked and resolved.

To be most effective, software inspections need to be fully integrated into the software development process from a technical as well as from a management point of view. Figure 1 presents an example using a simplified version of the Vorgehensmodell (V-Model) [19]. The products are sufficiently generic, and are found in one form or another in most, if not all, development process models.

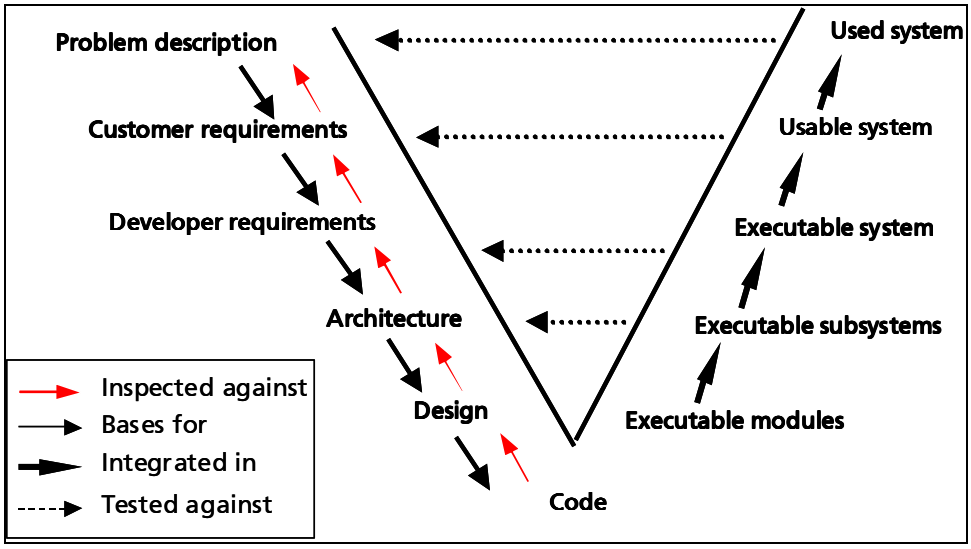

Figure 1: A Generic Software Development Model

The V-model is not a process model per se, but rather a product model since it does not define the sequence of development steps that must be followed to create the generic software development products. Hence, it is applicable with all process models for which products are developed in sequence, in parallel, or incrementally. The point is that the logical relationship between development products should be maintained and a software inspection can be triggered and scheduled as soon as a development product or parts of it are created. A typical example is a piece of code that is checked against the design document. This survey elaborates upon the question of how to set up and run inspections to achieve the stated goals, i.e., the achieve a certain level of effectiveness from a technical, economic, organizational, and tool perspective.

Related Work

Findings about inspections have not been easy to reconcile and consolidate due to the sheer volume of work already published. Hence, it is not surprising that the available surveys [70] [83] [106] [127] [135] only cover the most relevant published research in their reviews.

Broadly speaking, existing surveys can be summarized as follows. Kim et al. [70] present a framework for software development technical reviews including software inspection, Freedman and Weinberg's technical review [131], and Yourdon's walkthrough [140]. They segment the framework according to aims and benefits of reviews, human elements, review process, review outputs, and other matters. Macdonald et al. [83] describe the scope of support for the currently available inspection process and review tools. Porter et al. [106] focus their attention on the organizational attributes of the software inspection process, such as the team size or the number of sessions, to understand how these attributes influence the costs and benefits of software inspection. Wheeler et al. [135] discuss the software inspection process as a particular type of peer review process and elaborate the differences between software inspection, walkthroughs, and other peer review processes. Tjahjono [127] presents a framework for formal technical reviews (FTR) including objective, collaboration, roles, synchronicity, technique, and entry/exit-criteria as dimensions. Tjahjono's framework aims at determining the similarities and differences between the review process of different FTR methods, as well as identifying potential review success factors. All of these surveys contribute to the knowledge of software inspection by identifying factors that may impact inspection success. However, none of them presents its findings from a more global perspective. This makes it difficult for practitioners to determine which inspection method or refinement to choose, should they want to introduce inspection or improve on their current inspection approach.

Organization of this Survey

We organized this survey as follows. Section 2 presents the study methodology including our approach to identify and select the relevant software inspection literature. Section 3 describes the core concepts and relationships that together define the notion of software inspection in the form of a taxonomy. Section 4 discusses the most important literature sources in the context of the presented taxonomy. Section 5 concludes.

Study Methodology

Survey Motivation and Principles

Literature surveys have long played a central role in the accumulation of scientific knowledge. As science is a cumulative endeavor, any one theory or finding is suspect because of the large array of validity threats that must be ruled out. Moreover, all too often new techniques and methods are proposed and introduced, without building on the extensive body of knowledge that is incorporated in the ones already available. These problems can be somewhat alleviated by establishing the current facts using the mechanism of literature surveys. The facts are the dependable relationships among the various concepts that occur despite any biases that may be present in particular studies because of the implicit theories behind the investigators' choice of observations, measures, and instruments. Hence, a literature survey makes the implicit theories explicit by identifying their commonalities and differences, often from a specific angle when the body of knowledge has become very rich. In some cases, a literature survey may even be an impetus for the unification of existing theories to induce a new, more general theory that can be empirically tested afterwards.

To achieve these goals, a survey must fulfill several principles: First, it must be well-contained, that is, encapsulate its work within a clearly defined scope where the benefits of doing so can be well-understood and accepted. Second, a survey must provide profound breadth and depth regarding the literature relevant to its defined scope. Finally, it must present a unified vocabulary reconciling the most important terms in a field.

The first principle is the hardest to fulfill and illustrates the fact that there cannot be one single method for developing a survey since it is so tightly coupled with the notion of scope. The scope is, in fact, what defines the gist of a survey and hence, depending on the particular interest of the authors, a survey can be geared in different directions. This is clearly illustrated by the different directions taken by the five surveys we mentioned above. Each used a particular scope and rationale for its motivation.

To fulfill the second and third survey principles, finding and selecting the relevant literature is of utmost importance. We attempted to collect any publication fitting our definition of inspection, which captures, we believe, the essence of other definitions. However, no single method for locating relevant literature is perfect [28]. Hence, we utilized a combination of methods to locate articles and papers on our subject.

Sampling Approach

We conducted searches of the following two inspection libraries: Bill Brykczynski's collection of inspection literature [21] [136] and the Formal Technical Review Library [59]. To be sure not to miss a paper recently published, we performed three additional steps in search of inspection articles: First, we employed a keyword search in the INSPECT database of the OCLC [97] and the library of the Association of Computing Machinery [3] using the keyword "software inspection". Second, we manually searched the following journals published 1990 and 1998: IEEE Transactions on Software Engineering, IEEE Software, Journal of Systems & Software, Communications of the ACM, and ACM Software Engineering Notes. Finally, we looked at the reference sections of books dealing with software inspection [47] [124] and some conference proceedings. Table 1 shows the results of our literature search. The reader must keep in mind that some articles are cross-referenced among several libraries. We made the results of our literature search available on-line [44].

Source	Number of Articles
Literature in [136]	147
FTR-Library	204
OCLC Database	55
ACM Database	21
IEEE Transactions on Software Engineering	12
IEEE Software	9
Journal of Systems & Software	8
Communications of the ACM	4
ACM Software Engineering Notes	3
Other	17

Table 1: Summary of Search Results.

Considering the very large number of published articles available, it was impossible to give full attention to every article within this survey, although we carefully considered each and every one of them. We excluded articles based on the following rules: (a) the article is an opinion paper and, therefore, does not

represent tangible inspection experiences, (b) it takes considerable effort (money or time) to get an article, (c) one or several authors published several papers about similar work in journals and conference proceedings -- in this case, we considered the most relevant journal publication --, (d) an article does only provide a weak research or practical contribution, although we acknowledge the subjectivity of this criteria. However, we avoided the dangers of ignoring papers because they do not fit neatly into our taxonomy. When in doubt, we included them. Overall, we included a total of 120 articles and reports about software inspection in this survey.

Although we consider the selected sample of papers as representative of the work in the inspection area, we are aware that the published papers are only a biased sample of inspection work actually carried out in reality. There are two principal reasons for this, which we can only be aware of, without any hope of overcoming them:

- The "File drawer problem" - unpublished as well as unretrievable null results stored away by unknown researchers [114]. When inspections are unsuccessfully applied, they are most often not reported in the literature. In all the articles we reviewed, there is only one, which shows that inspection did not have the expected benefits [122]. Yet, we believe that there might be more unsuccessful inspection trials.
- The successful use of inspection might also be only sporadically reported, since that may reveal defect information unpalatable to companies engaged in competitive industries [1].

A Taxonomy for Inspection Approaches

Based on the selected literature, we derived a taxonomy to articulate the core concepts and relationships of software inspection. This taxonomy is organized around four primary dimensions -- technical, economic, organizational, and tools. With them, we characterized the nature of software inspection. For each primary dimension, we used a selection criterion in the form of a concrete goal to elicit from the relevant literature. Yet, though necessary, these four primary dimensions are not unique. They are relevant to the major areas of software development. Hence, we elicited from the literature particular sub-dimensions that we saw as fundamental to the nature and application of software inspection. Figure 2 shows the elicited dimensions and sub-dimensions.

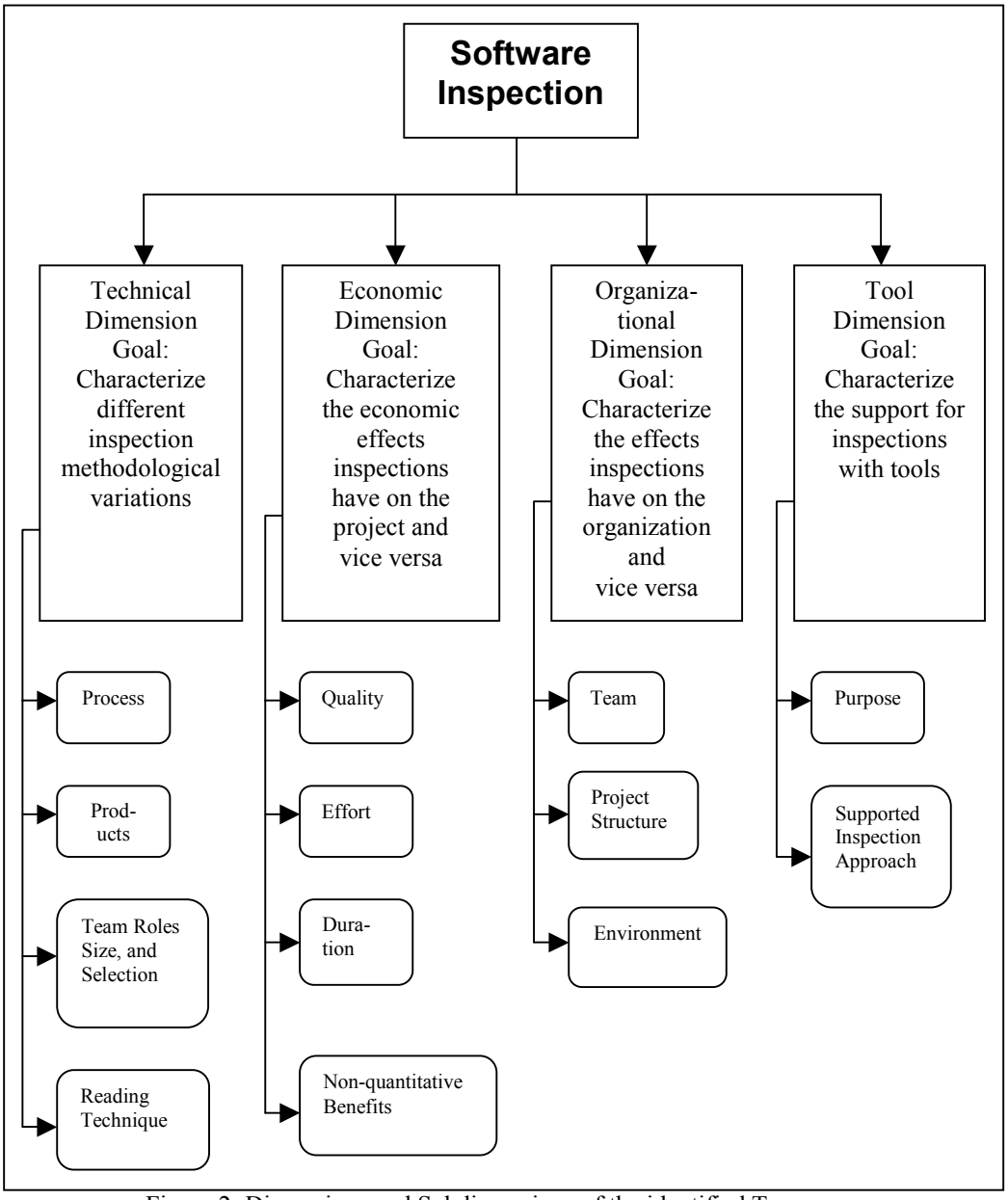

Figure 2: Dimensions and Subdimensions of the identified Taxonomy.

We briefly describe below each dimension and its associated primary goals. The major goal of the technical dimension is to characterize the different inspection methods so as to identify similarities and differences among them. For this, each inspection approach needs to be characterized in more detail according to the

activities performed (process), the inspected software product (product), the different team roles as well as overall optimal size and selection (team roles, size, and selection), and the technique applied to detect defects in the software product (reading technique)[2]. The economic dimension provides information on the effects that inspections have on a project and vice versa. This information is primarily relevant for decision makers or managers. They are most often interested in the way inspections influence product quality (quality), project effort (effort), and project duration (duration). However, inspections might also have other effects that a manager might be interested in, such as their contribution to team building or education in a particular project (others). The organizational dimension characterizes the effects inspections have on the whole organization and vice versa. For the organizational dimension, we elicited team (team), project structure (project structure), and environment (environment) as particular subdimensions. These subdimensions provide important information on the context in which inspections take place. Finally, the tool dimension describes how tools can support software inspections. For this dimension, we elicited the purpose of the various tools (purpose) and investigated how they support a given inspection approach (supported inspection approach).

We have to state that the dimensions are not completely orthogonal, that is, one dimension may be related to another dimension, but this is unavoidable. For example, a manager might base his/her decision about introducing inspections on the cost/benefit ratio inspections had in previous projects. Yet, we have done our best to minimize such overlap.

We now proceed by discussing in details each of Figure 2's dimensions and subdimensions using the relevant articles.

Discussion of published Work in the Context of the Taxonomy

The Technical Dimension of Software Inspection

Inspections must be tailored to fit particular development situations. To do this, it is essential to characterize the technical dimension of current inspection methods and their refinements to grasp the similarities and differences among them. As depicted in Figure 3, the technical dimension of our taxonomy includes the inspection process, the inspected product, the team roles participants have in an inspection as well as the team size, and the reading technique as subdimensions. Each of the subdimensions is discussed in more detail in this section. In total, we have identified 64 references relevant to this dimension.

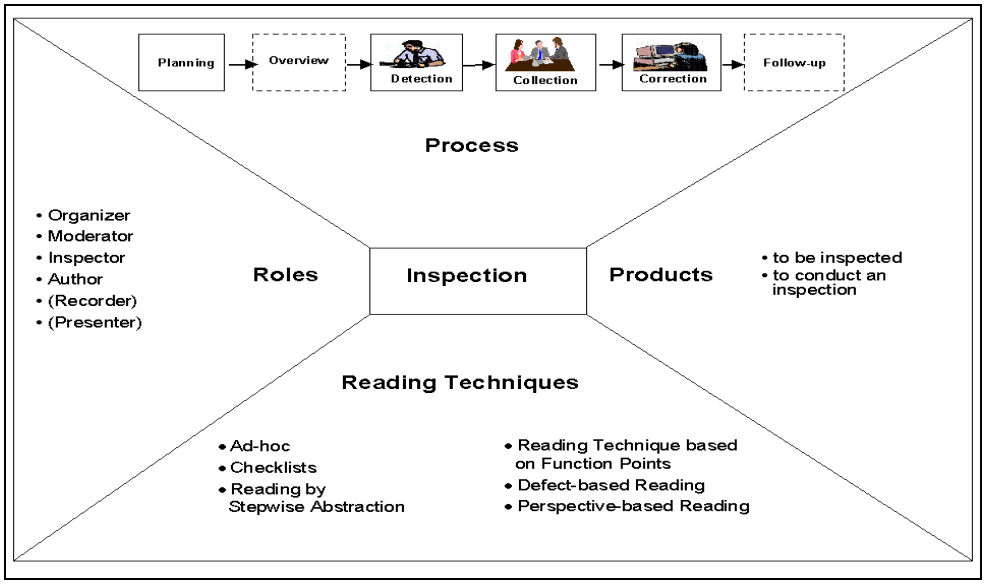

Figure 3: The Technical Dimension of Software Inspection.

Process

To explain the various similarities and differences among the inspection methods, a reference model for software inspection processes is needed. To define such a reference model, we adhered to the purpose of the various activities within an inspection rather than their organization. This allows us to provide an unbiased examination of the different approaches. We identified six major process phases: Planning, Overview, Defect Detection, Defect Collection, Defect Correction, and Follow-up. These phases can be found in many inspection methods or their

refinements. However, the question of how each phase is organized and performed often distinguishes one method from another.

Planning

The objective of the planning phase is to organize a particular inspection when materials to be inspected pass entry criteria, such as when source code successfully compiles without syntax errors. This phase includes the selection of inspection participants, their assignment to roles, the scheduling of the inspection meeting, and the partitioning and distribution of the inspection material. In most literature, this phase is not described in much detail, except in [1] [40] [47] [124]. However, we consider planning important to mention as a separate phase because there must be a person within a project or organization who is responsible for planning all inspection activities, even if such an individual plays numerous roles.

Overview

The overview phase consists of a first meeting in which the author explains the inspected product to other inspection participants. The main goal of the overview phase is to make the inspected product more lucid and, therefore, easier to understand and inspect for participants. Such a first meeting can be particularly valuable for the inspection of early artifacts, such as requirements or design documents, but also for complex source code. However, this meeting consumes effort and increases the duration of an inspection. Moreover, it may focus the attention of inspectors on particular issues, which prohibits an independent assessment of the inspected artifact. These limitations may be one reason why Fagan [40] states that an overview meeting for code inspection is not necessary. This statement is supported by Gilb and Graham [47]. They call the overview meeting the "Kickoff Meeting" and point out that such a meeting can be held, if desired, but is not compulsory for every inspection cycle. However, other authors consider this phase essential for effectively performing the subsequent inspection phases. Ackerman et al. [1], for example, argue that the overview brings all inspection participants to the point where they can easily read and analyze the inspected artifact. In fact, most published applications of inspections report performing an overview meeting [29] [34] [42] [43] [68] [71] [110] [111] [116] [125] [129] [134]. However, there are also examples that either did not perform one or did not report the performance of one [13] [73].

We found three conditions under which an overview meeting is definitely justified and beneficial. First, when the inspected artifact is complex and difficult to understand. In this case, explanations from the author about the inspected artifact facilitate the understanding of the inspected product for inspection participants. Second, if the inspected artifact belongs to a large software system, the author may want to explain the relationship between the inspected artifact and the whole software system to other participants. Third, new team members

participate in the inspection. Since they are inexperienced, explanations from the author put them in a position to inspect the artifact. In all three cases, explanations by the author may help other participants perform more effective inspection and save time in later inspection phases.

Defect Detection

The defect detection phase can be considered the core of an inspection. The main goal of the defect detection phase is to scrutinize a software artifact to identify defects. How to organize this phase is still debated in the literature. More specifically, the issue is whether defect detection is more an individual activity and hence should be performed individually, or whether defect detection is a group activity and should therefore be conducted as part of a group meeting, that is, an inspection meeting. Fagan [40] reports that a group meeting provides a synergy effect, that is, most of the defects are detected because inspection participants meet and scrutinize the inspection artifact together. He makes the implicit assumption that interaction contributes something to an inspection that is more than the mere combination of individual results. Fagan refers to this effect as the "phantom" inspector. However, others found little synergy in an inspection meeting. The most cited reference for this position is a paper by L. Votta [130]. His position is empirically supported in [117].

In many cases, authors distinguish between a "preparation" phase of an inspection, which is performed individually, and a "meeting" phase of an inspection, which is performed within a group [1] [40] [46] [56] [124]. However, it often remains unclear whether the preparation phase is performed with the goal detecting defects or just understanding the inspected artifact to detect defects later on in a meeting phase. For example, Ackerman et al. [1] state that a preparation phase lets the inspectors thoroughly understand the inspected artifact. They do not explicitly state that the goal of the preparation phase is defect detection. Bisant and Lyle [9] consider individual preparation as the vehicle for education. Other examples mention that the inspected artifact should be individually studied in detail throughout a preparation phase, but do not explicitly state education as a goal per se [25] [34] [42] [79].

Since the literature on software inspection does not provide a definite answer on which alternative to choose, we looked at some literature from the psychology of small group behavior [33] [80] [121]. Psychologists found that an answer to the question whether individuals or groups are more effective, depends upon the past experience of the persons involved, the kind of task they are attempting to complete, the process that is being investigated, and the measure of effectiveness. Since at least some of these parameters vary in the context of a software inspection, we recommend that the defect detection activity be organized as both individual and group activity with a strong emphasis on the former. Individual defect detection with the explicit goal to look for defects that should be resolved before the document is approved ensures that inspectors are well prepared for all following inspection steps. This may require extra effort on the inspectors' behalf

since each of them has to understand and scrutinize the inspected document on an individual basis. However, the effort is justified because, if a group meeting is performed later on, each inspector can play an active role rather than hiding himself or herself in the group and, thus, make a significant contribution to the overall success of an inspection.

Knight and Myers [72] [73] suggested the Phased Inspection Method. The main idea behind Phased Inspections is for each inspection phase to be divided into several mini-inspections or phases. Mini-inspections are conducted by one or more inspectors and are aimed at detecting defects of one particular class or type. This is the most important difference to "Conventional" inspections, which check for many classes or types of defects in a single examination. If there is more than one inspector, they will meet just to reconcile their defect list. The phases are done in sequence, that is, inspection does not progress to the next phase until rework has been completed on the previous phase.

Although Knight and Myers state that phased inspections are intended to be used on any work product, they only present some empirical evidence of the effectiveness of this approach for the code inspections. However, Porter et al. argue based on the results of their experiments [108], that multiple session inspections, that is, mini-inspections, with repair in between are not more effective for defec t detection but are more costly than conventional inspections. This may be one explanation why we did not find extensive use of the phased inspection approach in practice.

There has been noticeable growth in research on how individual defect detection takes place and can be supported with adequate techniques [5] [6] [109]. We tackle this issue later in more detail when we discuss reading techniques to support defect detection.

Defect Collection

In most published inspection processes, more than one person participates in an inspection and scrutinizes a software artifact for defects. Hence, the defects detected by each inspection participant must be collected and documented. Furthermore, a decision must be made whether a purported defect is really a defect. These are the main objectives of the defect collection phase. A further objective may be to decide whether the inspected artifact needs to be reinspected. The defect collection phase is most often performed in a group meeting. There, the decision as to whether or not a defect is really a defect is often a group decision. The same holds for the decision as to whether to perform a reinspection. This decision is often based on a subjective estimation of the defect content. A recent study revealed that this estimate can be quite accurate [39]. To make the reinspection decision more objective, some authors suggest to apply statistical models, such as capture-recapture models, for estimating the remaining number of defects in the software product after inspection [2] [22] [38] [115] [137]. If the estimate exceeds a certain threshold, the software product needs to be reinspected. However, recent studies [14] [15] showed that statistical estimators

are not very accurate for inspections with less than four inspection participants. Further research is necessary to validate this finding. In addition to statistical estimation models, graphical defect content estimation approaches are currently being investigated [138].

Since a group meeting is effort consuming and increases the development schedule, some authors suggest that such a meeting for inspections be abandoned. Instead, they offer the following alternatives [130]: Managed meetings, depositions, and correspondence. Managed meetings are well-structured meetings with a limited number of participants. A deposition is a three-person meeting in which the author, a moderator, and an inspector collect the inspectors' findings and comments. Correspondence includes forms of communication where the inspections and author never actually meet (e.g., by using electronic mail). Some researchers have elaborated on these alternatives [63]. Sauer et al. [118], for example, provide some theoretical underpinning for depositions. They suggest that the most experienced inspectors collect the defects and decide upon whether these are real or not.

In general, research does not seem to provide a conclusive answer to the question of whether inspection meetings pay off. We recommend that practitioners start with the "traditional" meeting-based approach and try later on whether non-meeting based approaches provide equivalent benefits. Regarding the benefits of group meetings, they also provide more intangible benefits such as dissemination of product information, development experiences, or enhancement of team spirit as reported in [30] [43]. Although difficult to measure, these benefits must be taken into account when a particular inspection approach is evaluated in addition to the number of defects it helps detect and remove.

On the other hand, these meetings are not problem-solving sessions. Neither personal conflicts among people or departments nor radically alternate solutions -- complete rewrite or redesign -- of the inspected artifact should be discussed there.

Defect Correction

Throughout the defect correction phase, the author reworks and resolves defects found [40] or rationalizes their existence [122]. For this he or she edits the material and deals with each reported defect. There is only little discussion in the literature about this activity [47] [124].

Follow-up

The objective of the follow-up phase is to check whether the author has resolved all defects. For this, one of the inspection participants verifies the defect resolution. Doolan reports that the moderator checks that the author has taken some remedial action for each defect detected [34]. However, others do not report a follow-up phase [95] [116] [122]. They either did not perform one or did not

consider it important. Furthermore, many consider the follow-up phase optional like the overview phase.

Products

The product dimension refers to the type of product that is usually inspected. The term product refers not only to the final delivered software system, but also to any documentation produced in the context of a software development project. Barry Boehm [12] stated that one of the most prevalent and costly mistakes made in software projects today is deferring the activity of detecting and correcting software problems until late in the project. This statement supports the use of software inspection for early life-cycle documents. However, a look at the literature reveals that in most cases inspection was applied to code documents. Figure 4 depicts how inspection was applied to various software products phasewise[3]. We found 20, 32, 55, and 12 papers that talk about the inspection of requirements, design, code, and testcase documents, respectively. These numbers demonstrate that the use of inspection is biased towards code documents. Although code inspection improves the code quality and provides savings, the savings are higher for early life-cycle artifacts as shown in a recent study [16], which integrates published inspection results into a coherent cost/benefit-model. The results of the study reveal that the introduction of code inspection saves 39 percent of defect costs compared to testing alone. The introduction of design inspection saves 44 percent of defect costs compared to testing alone. These findings motivate the use of inspections especially throughout the early development phases.

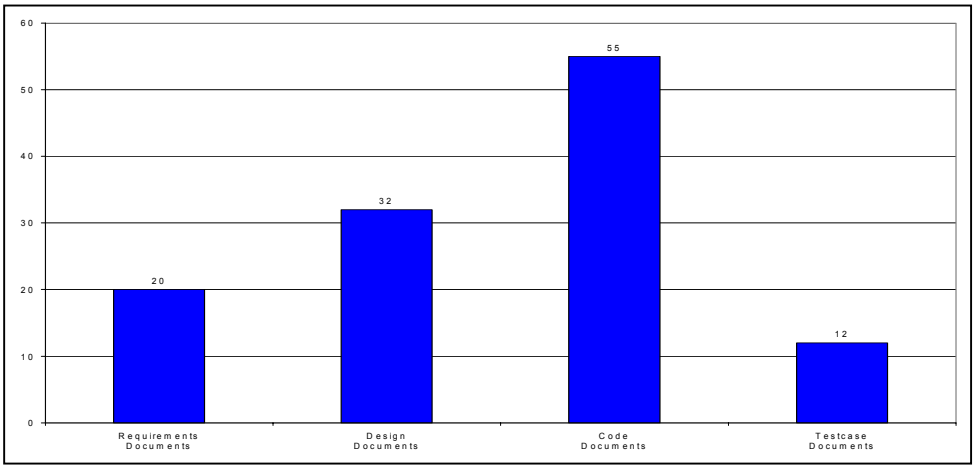

Figure 4: Number of References for the Inspection of various Software Development Products.

[3] We should note that some articles describe software inspection for several products. This explains why we only included 119 articles in this diagram.

The product dimension is also influenced by the development approach. Software inspections have been primarily used with products resulting from conventional structured development processes. Object-oriented products, particularly of the graphical form, have so far not been adequately addressed by inspection methods. Although some work in the area of object-oriented inspections exists [35] [128], there is a general lack of research regarding how the key features of the object-oriented paradigm impact software inspections.

Team Roles, Size, and Selection

Team Roles

Three important questions practitioners usually have about roles in software inspections are (1) what roles are involved in an inspection, (2) how many people are assigned to each role, and (3) how should people be selected for each roles. For the first question, a number of specific roles are assigned to inspection participants. Hence, each inspection participant has a clear and specific responsibility. The roles and their responsibilities are described in [1][40][116]. There is not much disagreement regarding the definition of inspection roles. In the following, we describe each of these roles in more detail:

- Organizer
  The organizer plans all inspection activities within a project or even across projects.
- Moderator
  The moderator ensures that inspection procedures are followed and that team members perform their responsibilities for each phase. He or she moderates the inspection meeting if there is one. In this case, the moderator is the key person in a successful inspection as he or she manages the inspection team and must offer leadership. Special training for this role is suggested.
- Inspector
  Inspectors are responsible for detecting defects in the target software product. Usually all team members can be assumed to be inspectors, regardless of their other roles in the inspection team.
- Reader/Presenter
  If an inspection meeting is performed, the reader will present the inspected products at an appropriate pace and lead the team through the material in a complete and logical fashion. The material should be paraphrased at a suitable rate for detailed examination. Paraphrasing means that the reader should explain and interpret the material rather than reading it literally.
- Author
  The author has developed the inspected product and is responsible for the correction of defects during rework. During an inspection meeting, he or

she addresses specific questions the reader is not able to answer. The author must not serve as moderator, reader, or recorder.

- Recorder
  The recorder is responsible for logging all defects in an inspection defect list during the inspection meeting.
- Collector
  The collector collects the defects found by the inspectors if there is no inspection meeting.

Team Size

To answer the second question, that is, how to assign resources to these roles in an optimal manner, the reported numbers in the literature are not uniform. Fagan recommends to keep the inspection team small, that is, four people [40] and Bisant and Lyle [9] have found performance advantages in an experiment with two persons: one inspector and the author, who can also be regarded as an inspector. Kusumoto et al. recently investigated the two-person approach in an educational environment [75]. Weller presents some data from a field study using three to four inspectors [133]. Madachy presents data showing that the optimal size is between three and five people [87]. Bourgeois corroborates these results in a different study [13]. Porter et al.'s experimental results suggest that reducing the number of inspectors from 4 to 2 may significantly reduce effort without increasing inspection interval or reducing effectiveness [107].

Martin et al. proposed the N-fold inspection method [68] [89] [119]. This inspection method is based on the hypothesis that a single inspection team can find only a fraction of the defects in a software product and that multiple teams will not significantly duplicate each others efforts. In an N-fold inspection, N teams each carry out parallel independent inspections of the same software artifact. In a sense, N-fold inspection scales up some ideas of scenario-based reading techniques, which are applied in the conventional inspection approach on an individual level, to a team level. The inspection participants of each independent inspection follow the various inspection steps of a conventional inspection as outlined previously, that is, individual defect detection with an Ad-hoc reading technique and defect collection in a meeting. The N-Fold inspection approach ends with a final step in which the results of each inspection team are merged into one defect list. It has been hypothesized that N different teams will detect more defects than a single large inspection team. In fact, there is already empirical evidence confirming this hypothesis [129]. However, if N independent teams inspect one particular document, inspection cost will be high. This limits this inspection approach to the inspection of early life-cycle artifacts for which very high quality really does matter, such as the aircraft industry, or safety critical systems [129].

We assume that there is no definite answer to the optimal number of inspectors and teamsize. An answer rather depends on the type of product and the environment in which an inspection is performed and the costs associated with

defect detection and correction in later development phases. However, in absence of a clear answer, we recommend starting with one team that consists of three to four people: One author, one or two inspectors, and one moderator (also playing the role of the presenter and scribe. After a few inspections, the benefits of adding an additional inspector or an additional team can be empirically evaluated. The evaluation involves an examination whether an additional person or team helps detect more defects, i.e., leads to an increase in inspection effectiveness and defect coverage. Of course, one must also address the question whether the effort for the extra person or team really pays off.

Team Selection

The final question is how to select members of an inspection team. Primary candidates for the role of inspectors are personnel involved in product development [41]. Outside inspectors may be brought in when they have a particular expertise that would add to the inspection [96]. Inspectors should have good experience and knowledge[4] [10] [41] [124]. However, the selection of inspectors according to experience and knowledge has two major implications. First, inspection results heavily depend upon human factors. This often limits the pool of relevant inspectors to a few developers working on similar or interfacing products [1]. Second, personnel with little experience are not chosen as inspectors although they may learn and, thus, profit a lot from inspection. Defect detection, that is, reading techniques, which we discuss later on in more detail, may alleviate these problems.

It is sometimes recommended that managers should neither participate nor attend an inspection [68] [96]. This stems from the fact that inspections should be used to assess the quality of the software product, not the quality of the people who create the product [41]. Using inspection data to evaluate people may result in less than honest and thorough inspections, since inspectors may be reluctant to identify defects if finding them results in a poor performance evaluation for a colleague.

Reading Technique

Recent empirical studies demonstrate that defect detection is more an individual than a group activity as assumed by many inspection methods and refinements [103] [130]. Moreover, these empirical studies show that a particular organization of the inspection process does not explain most of the variation in inspection results. Rather, one expects that inspection results depend on inspection participants themselves [108] and their strategies for understanding the inspected artifact [113]. Therefore, supporting inspection participants, that is, inspectors, with particular techniques that help them detect defects in software products, may

[4] In most cases, the kind of experience and knowledge is not well-defined in the articles.

increase the effectiveness of an inspection team most. We refer to such techniques as reading techniques.

A reading technique can be defined as a series of steps or procedures whose purpose is to guide an inspector in acquiring a deep understanding of the inspected software product. The comprehension of inspected software products is a prerequisite for detecting subtle and/or complex defects, those often causing the most problems if detected in later life-cycle phases. In a sense, a reading technique can be regarded as a mechanism or strategy for the individual inspector to detect defects in the inspected product. Of course, whether inspectors take advantage of this mechanism or strategy is up to them. At least, it is intended that inspectors use the available reading techniques since this makes the result of the defect detection activity less dependent on human factors, such as experience.

Even though reading is one of the key activities for individual defect detection [6], few documented reading techniques are currently available to support the activity. We found that ad-hoc reading and checklist-based reading are the most popular reading techniques used today for defect detection in inspection [40] [47].

Ad-hoc reading, by nature, offers very little reading support at all since a software product is simply given to inspectors without any direction or guidelines on how to proceed through it and what to look for. However, ad-hoc does not mean that inspection participants do not scrutinize the inspected product systematically. The word `ad-hoc' only refers to the fact that no technical support is given to them for the problem of how to detect defects in a software artifact. In this case, defect detection fully depends on the skill, the knowledge, and the experience of an inspector. Training sessions in program comprehension as presented in [113] may help subjects develop some of these capabilities to alleviate the lack of reading support. Although an ad-hoc reading approach was only mentioned a few times in the literature [34] [122], we found many articles in which little was mentioned about how an inspector should proceed in order to detect defects. Hence, we assumed that in most of these cases no particular reading technique was provided because otherwise it would have been stated.

Checklists offer stronger, boilerplate support in the form of questions inspectors are to answer while reading the document. These questions concern quality aspects of the document. Checklists are advocated in more than 25 articles. See, for example, [1] [40] [41] [54] [126], and Gilb and Grahams' manuscript [47]. Although reading support in the form of a list of questions is better than none (such as ad-hoc), checklist-based reading has several weaknesses. First, the questions are often general and not sufficiently tailored to a particular development environment. A prominent example is the following question: “Is the inspected artifact correct?” Although this checklist question provides a general framework for an inspector on what to check, it does not tell him or her in a precise manner how to en-sure this quality attribute. In this way, the checklist provides little support for an inspector to understand the inspected artifact. But this can be vital to detect major application logic defects. Second, concrete

instructions on how to use a checklist are often missing, that is, it is often unclear when and based on what information an inspector is to answer a particular checklist question. This weakness becomes apparent when looking at the example presented above. In fact, several strategies are actually feasible to address all the questions in a checklist. The following approach characterizes the one end of the spectrum: The inspector takes a single question, goes through the whole artifact, answers the question, and takes the next question. The other end is defined by the following procedure: The inspector reads the document. Afterwards he or she answers the questions of the checklist. It is quite unclear which approach inspectors follow when using a checklist and how they achieved their results in terms of defects detected. The final weakness of a checklist is the fact that checklist questions are often limited to the detection of defects that belong to particular defect types. Since the defect types are based on past defect information [24], inspectors may not focus on defect types not previously detected and, therefore, may miss whole classes of defects.
To address some of the presented difficulties, one can develop a checklist according to the following principles:

- The length of a checklist should not exceed one page.
- The checklist question should be phrased as precise as possible.
- The checklist should be structured so that the quality attribute is clear to the inspector and the question give hints on how to assure the quality attribute.

In some cases the length of a checklist may exceed one page. In these cases, it may be possible to make inspectors responsible for different parts of the checklist. Although these actions can be taken, a checklist still provides very little guidance for inspectors on how the perform the various checks. This weakness led to the development of more procedural reading techniques.
Techniques providing more structured and precise reading instructions include both a reading technique denoted "Reading by Stepwise Abstraction" for code documents advocated by the Cleanroom community [36] [37] [81], as well as a technique suggested by Parnas et. al. called Active Design Review [98] [99] for the inspection of design documents.

The technique “Reading by Stepwise Abstraction” was described in the context of the Verification-based Inspection approach [37]. This is an inspection variation used in conjunction with the Cleanroom software development method. Although this method requires the author(s) to perform various inspections of work products, the inspection process itself is not well described in the literature. We found that it consists of at least one step, in which individual inspectors examine the work product using "Reading by Stepwise Abstraction". This technique requires an inspector to read a sequence of statements in the code and to abstract the functions these statements compute. An inspector repeats this procedure until the final function of the inspected code artifact has been abstracted and can be compared to the specification. This reading technique is

limited to code artifact, though it provides a more formal approach for inspectors to check the functional correctness [37]. We found little information on the inspection process after the individual defect detection step. However, the Cleanroom approach is one of the few development approaches in which defect detection and inspection activities are tightly integrated in and coupled with development activities. The Cleanroom approach and its integrated inspection approach have been applied to several development projects [6] [31] [36].

Parnas and Weiss suggest an inspection method denoted as Active Design Reviews (ADR) for inspecting design documents [98] [99]. The authors believe that in conventional design inspections, inspectors are given too much information to examine, and that they must participate in large meetings, which only allow for limited interaction between inspectors and author. To tackle theses issues inspectors are chosen based on their specific level of expertise skills and assigned to ensure thorough coverage of design documents. Only two roles are defined within the ADR process. An inspector has the expected responsibility of finding defects, while the designer is the author of the design being scrutinized. There is no indication of who is responsible for setting up and coordinating the review. The ADR process consists of three steps. It begins with an overview step, where the designer presents an overview of the design and meeting times are set. The next step is the defect detection step for which the author provides questionnaires to guide the inspectors. The questions are designed such that they can only be answered by careful study of the design document, that is, inspectors have to elaborate the answer instead of stating yes/no. Some of the questions reinforce an active inspection role by making assertions about design decisions. For example, he or she may be asked to write a program segment to implement a particular design in a low-level design document being inspected. The final step is defect collection, which is performed in inspection meetings. However, each inspection meeting is broken up into several smaller, specialized meetings, each of which concentrates on one quality property of the artifact. An example is checking consistency between assumptions and functions, that is, determining whether assumptions are consistent and detailed enough to ensure that functions can be correctly implemented and used.
Active Design Review is an important inspection variation because ADR inspectors are guided by a series of questions posed by the author(s) of the design in order to encourage a thorough defect detection step. Thus, inspectors get reading support when scrutinizing a design document. Although little empirical evidence shows the effectiveness of this approach, other researchers based their inspection variations upon these ideas [23] [72].

A more recent development in the area of reading techniques for individual defect detection in software inspection is Scenario-based reading [6]. The essence of the Scenario-based reading idea is the use of the notion of scenarios that provide custom guidance for inspectors on how to detect defects. A scenario may

be a set of questions or a more detailed description for an inspector on how to perform the document review. Principally, a scenario limits the attention of an inspector to the detection of particular defects as defined by the custom guidance. Since each inspector may use a different scenario, and each scenario focuses on different defect types, it is expected that the inspection team, together, becomes more effective. Hence, it is clear that the effectiveness of a scenario-based reading technique depends on the content and design of the scenarios. So far, researchers have suggested three different approaches for developing scenarios and, therefore, three different scenario-based reading techniques: Defect-based Reading [109] for inspecting requirements documents, a scenario-based reading technique based on function points for inspecting requirements documents [23], and Perspective-based Reading for inspecting requirements documents [5] or code documents [76].

The main idea behind Defect-based Reading is for different inspectors to focus on different defect classes while scrutinizing a requirements document [93] [105] [109] [117]. For each defect class, there is a scenario consisting of a set of questions an inspector has to answer while reading. Answering the questions helps an inspector primarily detect defects of that particular class. The defect-based reading technique has been validated in a controlled experiment with students as subjects. The major finding was that inspectors applying Defect-based Reading detect more defects than inspectors applying either Ad-hoc or checklist-based reading.

Cheng and Jeffery have chosen a slightly different approach to define scenarios for defect detection in requirements documents [23]. This approach is based on Function Point Analysis (FPA). FPA defines a software system in terms of its inputs, files, inquiries, and outputs. The scenarios, that is, the Function Point Scenarios, are developed around these items. A Function Point Scenario consists of questions and directs the focus of an inspector to a specific function-point item within the inspected requirements document. The researchers carried out an experiment to investigate the effectiveness of this approach compared to an ad-hoc approach. The experimental results show that, on average, inspectors following the ad-hoc approach found more defects than inspectors following the function-point scenarios. However, experience seemed to be a confounding factor that biased the results of the experiment.

The main idea behind the perspective-based reading technique is that a software product should be inspected from the perspective of different stakeholders [5] [18] [76] [77]. The rationale is that there is no single monolithic definition of software quality, and little general agreement about how to define any of the key quality properties, such as correctness, maintainability, or testability. Therefore, inspectors of an inspection team have to check software quality as well as the software quality factors of a software artifact from different perspectives. The perspectives mainly depend upon the roles people have within the software development or maintenance process. For each perspective, either one or multiple scenarios are defined, consisting of repeatable activities an inspector has to

perform, and questions an inspector has to answer. The activities are typical for the role within the software development or maintenance process, and help an inspector increase his or her understanding of the software product from the particular perspective. For example, designing test cases is a typical activity performed by a tester. Therefore, an inspector reading from the perspective of a tester may have to think about designing test cases to gain an understanding of the software product from the tester's point of view. Once understanding is achieved, questions about an activity or questions about the result of an activity can help an inspector identify defects.

Reading a document from different perspectives is not a completely new idea. It was seeded in early articles on software inspection, but never worked out in detail. Fagan [40] reports that a piece of code should be inspected by the real tester. Fowler [42] suggests that each inspection participant should take a particular point of view when examining the work product. Graden et al. [49] state that each inspector must denote the perspective (customer, requirements, design, test, maintenance) by which they have evaluated the deliverable. So far, the perspective-based reading technique has been applied to inspecting requirements documents [5], object-oriented design models [77], and code documents [76].

General prescriptions about which reading technique to use in which circumstances can rarely be given. However, in order to compare them, we set up the following criteria: Application Context, Usability, Repeatability, Adaptability, Coverage, Training, and Validation. The criteria are to provide answers to the following questions:

- Application Context: To which software products can a reading technique be applied and to which software products has a reading technique already been applied?
- Usability: Does a reading technique provide prescriptive guidelines on how to scrutinize a software product for defects?
- Repeatability: Are the results of an inspector's work repeatable, that is, are the results such as the detected defects, independent of the person looking for defects?
- Adaptability: Is a reading technique adaptable to particular aspects, e.g., notation of the document, or typical defect profiles in an environment?
- Coverage: Are all required quality properties of the software product, such as correctness or completeness, verified in an inspection?
- Training required: Does the reading technique require some training on the inspectors' behalf?
- Validation: How was the reading technique validated, that is, how broadly has it been applied so far?

Table 2 below characterizes each reading technique according to these criteria. We use question marks in cases for which no clear answer can be provided.

Reading Technique	**Characteristics**						
	Application Context	Usability	Repeatability	Adaptability	Coverage	Training required	Validation
Ad-hoc	All Products	No	No	No	No	No	Industrial Practice
Checklists	All Products	No	No	Yes	Case Dependent	No	Industrial Practice
Reading by stepwise Abstraction	All Products allowing abstraction, Functional Code	Yes	Yes	No	High for correctness defects	Yes	Applied primarily in Cleanroom projects
Active Design Reviews	Design, Design	Yes	Yes	Yes	?	?	Initial Case Study
Defect-based reading	All Products, Requirements	Yes	Case Dependent	Yes	High	Yes	Experimental Validation
Reading based on function points	All Products, Requirements	Yes	Case Dependent	Yes	?	Yes	Experimental Validation
Perspective-based reading	All Products , Requirements, Design, Code	Yes	Yes	Yes	High	Yes	Experimental Validation and initial Industrial Use

Table 2: Characterization of Reading Techniques.

The Economic Dimension of Software Inspection

One of the most important criteria for choosing a particular inspection approach is the effort a particular inspection method or refinement consumes. Effort is an issue in which project managers are mainly interested. Hence, we refer to this dimension as the managerial dimension. To make a sound evaluation, that is, to determine whether it is worth spending effort on inspection, one must also consider how inspections affect the quality of the software product as well as the cost and the duration of the project in which they are applied. We discuss a sample of 24 articles in the context of these three subdimensions.

In the context of a software development project, the investigation of inspection benefits is usually done in a separate analysis phase after one or several inspections have been performed. The analysis phase, therefore, does not directly belong to the inspection process. It rather represents a kind of characterization or improvement process in addition to inspections.

Quality

Some authors state that inspections can reduce the number of defects reaching testing by ten times [45]. However, these statements are often based on personal opinion rather than on collected inspection data. Hence, we focus our discussion about quality on examples of published inspection data taken from the literature. We emphasize that many of the data reported in the literature are not presented in a manner that allows straightforward comparison and analysis as pointed out by [16]. Table 3 summarizes the results of our analysis. It demonstrates the current trend that inspections help detect defects and, thus, improve the quality of the inspected documents. We need to emphasize that a direct comparison of inspection results between environments should not be done, e.g., one company is not "better" than another one just because the defect detection effectiveness is higher.

Reference	Environment	Result
Fagan [40][41]	Aetna Life Casuality	38 defects from 46 detected
	IBM Respond, United Kingdom	93% of all defects were detected by inspections
	Standard Bank of South Africa	Over 50% of all defects detected by inspection
Weller [132]	Bull HN Information Systems	70% of all defect detected by inspection
Grady and van Slack [51]	Hewlett-Packard	60%-70% of all defects detected by inspection
Shirey [122]		60%-70% of all defects detected by inspection
Barnard and Price [4]	AT&T Bell Laboratories	30%-75% of all defects detected by inspection
McGibbon [92]	Cardiac Pacemakers Inc.	70% to 90% of all defects detected by inspection
Collofello and Woodfield [26]	Large real time software project	Defect detection effectiveness is 54% for design inspection, 64% for code inspection, and 38% for testing
Kitchenham et al. [71]	ICL	57.7% of all defects found by code inspection
Franz and Shih [43]	Hewlett Packard	19% of all defects found by inspection
A. Gately [46]	Raytheon Systems Company	The average number of defects found by inspection is 18.2.
Conradi et al. [27]	Ericsson	The average number of defects found by inspection is 3.41.

Table 3: Summary of Inspection Results with respect to Quality.

Fagan [40] presents data from a development project at Aetna Life and Casualty. Two programmers have written an application program of eight modules (4439 non-commentary source statements) in Cobol. Design and code inspections were introduced into the development process. After 6 months of actual usage, 46 defects had been detected during development and usage of the program. Fagan reports that 38 defects had been detected by design and code inspections together, yielding a defect detection effectiveness for inspections of 82%. In this case, the defect detection effectiveness was defined as the ratio of defects found and the total number of defects in the inspected software product. The remaining 8 defects had been found during unit test and preparation for acceptance test. In another article, Fagan[41] publishes data from a project at IBM Respond, United Kingdom. Seven programers developed a program of 6271 LOC in PL/1. Over the life cycle of the product, 93% of all defects were detected by inspections. He also mentions two projects of the Standard Bank of South Africa (143 KLOC) and American Express (13 KLOC of system code), each with a defect detection effectiveness for inspections of over 50% without using trained inspection moderators.

Weller [132] presents data from a project at Bull HN Information Systems, which replaced inefficient C code for a control microprocessor with Forth. After system tests had been completed, code inspection effectiveness was around 70%. Grady and van Slack [51] report on experiences from achieving widespread inspection use at HP. In one of the company's divisions, inspections (focusing on code) typically found 60 to 70% of the defects. Shirey [122] states that defect detection effectiveness of inspections is typically reported to range from 60 to 70%. Barnard and Price [4] cite several references and report a defect detection effectiveness for code inspections varying from 30% to 75%. In their environment at AT&T Bell Laboratories, the authors achieved a defect detection effectiveness for code inspections of more than 70%. McGibbon [92] presents data from Cardiac Pacemakers Inc. where inspections are used to improve the quality of life-critical software. They observed that inspections removed 70 to 90% of all faults detected during development. Collofello and Woodfield [26] evaluated reliability-assurance techniques in a case study - a large real-time software project that consisted of about 700,000 lines of code developed by over 400 developers. The respective defect detection effectiveness is reported to be 54% for design inspections, 64% for code inspections, and 38% for testing. More recently, Raz and Yaung [110] presented the results of an analysis of defect-escape data from design inspection in two maintenance releases of a large software product. They found that the less effective inspections were those with the largest time investment, the likelihood of defect escapes being clearly affected by the way in which the time was invested and by the size of the work product inspected. Kitchenham et al. [71] report on experience at ICL, where 57.7% of defects were found by software inspections. The total proportion of development effort devoted to inspections was only 6%. Gilb and Graham [47] include experience data from various sources in their discussion of the benefits

and costs of inspections. IBM Rochester Labs publish values of 60% for source code inspections, 80% for inspections of pseudocode, and 88% for inspections of module and interface specifications. Grady [50] performs a cost/benefit analysis for different techniques, among them design and code inspections. He states that the average percentage of defects found for design inspections is 55%, and 60% for code inspections. Franz and Shih [43] present data from code inspection of a sales and inventory tracking systems project at HP. This was a batch system written in COBOL. Their data indicate that inspections had 19% effectiveness for defects that could also be found during testing. Myers [95] performed an experiment to compare program testing to code walkthroughs and inspections. This research is based on work performed earlier by Hetzel [53]. The subjects were 59 highly experienced data processing professionals testing and inspecting a PL/I program. Myers [95] reports an average effectiveness value of 38% for inspections. This controlled experiment was replicated several times [7] [66] [139] with similar results. A. Gately [46] presents some results from the Raytheon Systems Company. In this study, historical review data from a large real-time embedded system were analyzed. She found an average number of defects of 18.2 defects. Conradi et al. [27] present some review results from Ericsson. They studied two projects in which reviews and testing were used. The data comes from a site that passed CMM level 2 certification and aims for level 3 in the year 2000. They report an avarage number of defects of 3.41.

Effort

It is necessary for a project manager to have a precise understanding of the effort associated with inspections. Since inspection is a human-based activity, inspection costs are determined by human effort. The most important question addressed in literature is whether an inspection effort is worth making when compared to the effort for other defect detection activities, such as testing. Most of the literature present solid data supporting the claim that the costs for detecting and removing defects during inspections is much lower than detecting and removing the same defects in later phases. Table 4 summarizes the results of our analysis.

Reference	Environment	Result
Kaner [68]	Jet Propulsion Laboratory	Ratio of the cost of fixing defects during inspection to fixing them during formal testing range from 1:10 to 1:34
Remus [112]	IBM Santa Teresa Lab	Ratio of the cost of fixing defects during inspection to fixing them during formal testing is 1:20
Kan [67]	IBM Rochester Lab	Ratio of the cost of fixing defects during inspection to fixing them during formal testing is 1:13
Ackerman et al. [1]	Different Projects	0.58 – 5 hours per defect found (lower than in testing)
Weller [133]	Bull HN Information Systems	1.43 hours per defect in inspection and 6 hours in testing
Collofello and Woodfield [26]	Large real time software project	7.5 hours per defect for design inspection, 6.3 hours per defect for code inspection, and 11.6 hours per defect for testing
Franz and Shih [43]	Hewlett Packard	1 hour per defect for inspection and 6 hours per defect for testing
Kelly et al. [69]	Jet Propulsion Laboratory	1.75 hours per defect of design inspection, 1.46 hours per defect for code inspection, 17 hours per defect for testing
Kitchenham et al. [71]	ICL	1.58 hours per defect in design inspection
Gilb and Graham [47]	Applicon	0.9 hours to find and fix a major defect.
Bourgeois [13]	Lockheed Martin Western Development Labs	1.3 hours per defect found and 1.4-1.8 hour per defect found and fixed

Table 4: Summary of Inspection Results with respect to Effort.

The Jet Propulsion Laboratory (JPL) found the ratio of the cost of fixing defects during inspections to fixing them during formal testing ranged from 1:10 to 1:34 [68], at the IBM Santa Teresa Lab the ratio was 1:20 [112], and at the IBM Rochester Lab it was 1:13 [67].

We must say that authors often relate the costs to either the size of the inspected product or the number of defects found. Ackerman et al. [1] present data on different projects as a sample of values from the literature and from private reports:

- The development group for a small warehouse-inventory system used inspections on detailed design and code. For detailed design, they reported 3.6 hours of individual preparation per thousand lines, 3.6 hours of meeting time per thousand lines, 1.0 hours per defect found, and 4.8 hours per major defect found (major defects are those that will affect execution). For source code, the results were 7.9 hours of preparation per thousand lines, 4.4 hours of meetings per thousand lines, and 1.2 hours per defect found.

- A major government-systems developer reported the following results from inspection of more than 562,000 lines of detailed design and 249,000 lines of source code: For detailed design, 5.76 hours of individual preparation per thousand lines, 4.54 hours of meetings per thousand lines, and 0.58 hours per defect found. For code, 4.91 hours of individual preparation per thousand lines, 3.32 hours of meetings per thousand lines, and 0.67 hours per defect found.
- Two quality engineers from a major government-systems contractor reported 3 to 5 staff-hours per major defect detected by inspections, showing a surprising consistency over different applications and programming languages.
- A banking computer-services firm found that it took 4.5 hours to eliminate a defect by unit testing compared to 2.2 hours by inspection (these were probably source code inspections).
- An operating-system development organization for a large mainframe manufacturer reported that the average effort involved in finding a design defect by inspections is 1.4 staff-hours compared to 8.5 staff-hours of effort to find a defect by testing.

Weller [133] reports data from a project that performed a conversion of C code to Fortran for several timing-critical routines. While testing the rewritten code, it took 6 hours per failure. It was known from a pilot project in the organization that they had been finding defects in inspections at a cost of 1.43 hours per defect. Thus, the team stopped testing and inspected the rewritten code, detecting defects at a cost of less than 1 hour per defect.

Collofello and Woodfield [26] estimate some factors for which they had insufficient data. They performed a survey among many of the 400 members of a large real-time software project who were asked to estimate the effort needed to detect and correct a defect for different techniques. The results were 7.5 hours for a design error, 6.3 hours for a code error, both detected by inspections, 11.6 hours for an error found during testing, and 13.5 hours for an error discovered in the field.

Franz and Shihs data [43] indicate that the average effort per defect for code inspections was 1 hour and for testing 6 hours. In presenting the results of analyzing inspections data at JPL, Kelly et al. [69] report that it takes up to 17 hours to fix defects during formal testing, based on a project at JPL. They also report approximately 1.75 hours to find and fix defects during design inspections, and approximately 1.46 hours during code inspections.

There are also examples that present findings from applying inspections only as a quality assurance activity. Kitchenham et al. [71], for instance, report on experience at ICL where the cost of finding a defect in design inspections was 1.58 hours.

Gilb and Graham [47] include experience data from various sources in their discussion of the benefits and costs of inspections. A senior software engineer describes how software inspections started at Applicon. In the first year, 9 code

inspections and 39 document inspections (documents other than code) were conducted and an average effort of 0.8 hours was spent to find and fix a major problem. After the second year, a total of 63 code inspections and 100 document inspections had been conducted and the average effort to find and fix a major problem was 0.9 hours.

Bourgeois [13] reports experience from a large maintenance program within Lockheed Martin Western Development Labs where software inspections replaced structured walkthroughs in a number of projects. The analyzed program was staffed by more than 75 engineers who maintain and enhance over 2 million lines of code. The average effort for 23 software inspections (6 participants) was 1.3 staff-hours per defect found and 2.7 staff-hours per defect found and fixed. Bourgeois also presents data from Jet Propulsion Laboratory, which is used as an industry standard. There, the average effort for 171 software inspections (5 inspection participants) was 1.1 staff-hours per defect found and 1.4 to 1.8 staff-hours per defect found and fixed.

Because inspection is a human-intensive activity and, therefore, effort consuming, managers are often critical or even reluctant to use them the first time. Part of the problem is the perception that software inspections cost more than they are worth. However, available quantitative evidence as presented above indicates that inspections have had significant positive impact on the quality of the developed software and that inspections are more cost-effective than other defect detection activities, such as testing. Furthermore, it is important to keep in mind that besides quality improvement and cost savings realized by finding and fixing defects before they reach the customer, other benefits are often associated with performing inspections. These benefits, such as learning, are often difficult to measure, but they also have an impact on quality, productivity, and the success of a software development project.

Duration

Inspections do not only consume effort, but they also have an impact on the product's development cycle time. Inspection activities are scheduled in a way in which all people involved can participate and fulfill their roles. Thus, the interval for the completion of all activities will range from at least a few days up to a few weeks. During this period, other work that relies on the inspected software product may be delayed. Hence, duration might be a crucial aspect for a project manager if time to market is a critical issue during development. However, only few articles present information on the global inspection duration.

Votta discusses the effects of time loss due to scheduling contention. He reports that inspection meetings account for 10 percent of the development interval [130]. Due to the delays, he advises substituting inspection meetings with other forms of defect collection.

The Organizational Dimension of Software Inspection

Fowler [42] states that the introduction of inspection is more than giving individuals the set of skills on how to perform inspections: It also introduces a new process within an organization. Hence, it affects the whole organization, that is, the team, the project structure, and the environment. We identified 6 references relevant to this dimension.

Team

An important factor regarding software inspection is the human factor. A Software inspection is driven by its participants, i.e., the members of a project team. Hence, the success or failure of software inspection as a tool for quality improvement and cost reduction heavily depends on human factors. If team members are unwilling to perform inspections, all efforts will be deemed to fail. Franz and Shih [43] point out that attitude about defects is the key to effective inspections. Once the inevitability of defects is accepted, team members often welcome inspections as a defect detection method. To overcome objections, Russell reports on an advertising campaign to persuade project teams that inspections really do work [116]. An advice we often found in the literature was to exclude management from an inspection [43] [68]. This is suggested to avoid any misconception that inspection results are used for personnel evaluation. Furthermore, training is deemed essential [1] [42]. Training allows project members to build their own opinion on how inspections work and how crucial defect data are within an environment for triggering further empirically justified process improvements.

Project Structure

Inspection per se is a human-based activity. Especially when meetings are performed, authors are confronted with the defects they created. This can easily result in personal conflicts, particularly in project environments with a strict hierarchy. Hence, one must consider the project structure to anticipate the conflict potential among participants. Depending on this potential for conflict, one must decide whether an inspection moderator belongs to the development team or must come from an independent department. This is vital in cases in which inspection is applied between sub-groups of one project. Personal conflicts within an inspection result in demotivation for performing inspection at all.

Environment

Introducing inspections is a technology transfer initiative. Hence, issues revolve around the need to deal with a software development organization, not just in terms of its workers but also in terms of its culture, management, budget, quality, and productivity goals. All these aspects can be subsumed in the subdimension environment of an organization. Fowler [42] states that preparing the organization for using inspections dovetails with adapting the inspections to the

local technical issues. Furthermore, the new process must be carefully designed to serve in the organization's environment and culture. Based on their inspection experiences at Hewlett-Packard, Grady and Van Slack [51] suggest a four stage process for inspection technology transfer: Experimental stage, initial guideline stage, widespread belief and adoption stage, and standardization stage. The experimental stage comprises the first inspection activities within an organization, and is often limited to a particular project of an organization. Based on the experiences in this project, first guidelines can be developed. This is the starting point for the initial guideline stage. In this stage, the inspection approach is defined in more detail and training material is created. The widespread belief and adoption stage takes advantage of the available experiences and training material to adopt inspection to several projects. Finally, the standardization stage helps build an infrastructure structure strong enough to achieve and hold inspection competence. This approach follows a typical new technology transfer model.

The Tool Dimension of Software Inspection

Currently, few tools supporting inspections are available. Some of them were developed by researchers to investigate software (often source code) inspection and none of the academic tools has reached commercial status yet. There may be some commercial tools available that we were not aware of, since they have not been discussed in the inspection literature. We analyzed, discussed, and classified the following ten inspection tools: (1) PAE (Program Assurance Environment) [8], which can be seen as an extended debugger and represents an exception in the list of tools. (2) InspecQ [73] concentrates on the support of the Phased Inspection process model developed by Knight and Meyers (3) ICICLE [20] supports the defect detection phase as well as the defect collection phase in a face-to-face meeting. (4) Scrutiny [48] and (5) CSI [90], support synchronous, distributed meetings to enable the inspection process for geographically separated development teams. (6) CSRS [62] (7) InspectA [73], (8) Hypercode [101], and (9) AISA [102] removes the conventional defect collection phase and replace it with a public discussion phase were participants vote on defect-annotations. (10) ASSIST [83] uses its own process modelling language and executes any desired inspection process model. All tools provide more or less comfortable document handling facilities for browsing documents on-line.

To compare the various tools, we developed Table 3 according to the various phases of the inspection process. For the defect detection phase we added a row to characterize the capability of a tool to automate the defect detection process (e.g., with rule sets). For the defect collection phase, we added a row to determine whether a tool supports defect collection in a synchronous (i.e. same time), asynchronous (i.e., different time), local (i.e., same place), or distributed (i.e., different place) manner. We focused on whether a tool provides facilities to control and measure the inspection process, and on the infrastructure on which the tool is running (a cross `x' indicates support and a minus `-' no support). Of

course, for source code products various compilers are available that can perform type and syntactical checking. This may remove some burden from inspectors. Furthermore, support tools, such as Lint for C, may help detect further classes of defects. However, the use of these tools is limited to particular development situations and may only lighten the inspection burden.

	PAE	ICICLE	Scruti-ny	CSRS	Ins-pecQ	ASSISST	CSI/ CAIS	Inspec-tA	Hyper-Code	AISA
Refer-ences										
Planning Support	-	-	-	X	X	X	-	X	X	-
Defect Detection Support	X	X	X	X	X	X	X	X	X	X
Auto-mated Defect Detection	X	X	-	-	X	-	-	-	-	-
Annota-tion Sup-port	-	X	X	X	X	X	X	X	X	x
Document Handling Support	C-Cod e	C-code	Code	Code/ Text	C-Code Ada	Code	Code	Code	Code/ Text	Code/ Text/ Graph
Reading Tech-nique	Che cklist	Checklist	-	Checkl ist	Checkl ist	-	Check list	Check-list	-	-
Defect Collection Support	-	X	X	X	-	X	X	X	X	X
(Synch/As ynch)/(Lo cal/Distrib uted)	-/- -/-	S/- L/-	S/- L/D	-/A -/D	-/- -/-	S/A I/D	S/A L/D	-/A -/D	S/A -/D	-/A -/D
Defect Correction Support	-	-	-	-	-	-	-	(x)	(x)	x
Inspection Process	-	-	-	X	X	X	-	X	X	X

Control possible										
Process Meas-urement Support	-	X	X	X	-	X	X	X	X	x
Defect Statistics	-	X	X	X	-	X	X	X	X	X
Supported Infrastruc-ture	Unix	Unix/X-Windows	ConvB	Ergret	?	LAN	Suite	E-Mail	WWW	WWW

Table 3: Overview of Inspection Tools.

We must admit that the question of how to support inspections with tools is addressed by many researchers and companies. Hence, we might have missed some tool that may be beneficial for an inspection.

Conclusions

This survey presented a survey of work in the area of software inspection. The survey introduced a detailed description of the core concepts and relationships that together define the field of software inspection technologies.
This type of survey is beneficial to researchers and practitioners for various reasons. First, it provides a roadmap in the form of a taxonomy that allows for the identification of available inspection methods and experience. Hence, this survey helps identify the ingredients of the best-suited inspection approach for a particular situation through the combination of the various dimensions. Second, the work helps structure the large amount of published inspection work. This structure presents the gist of the inspection work so far performed and helps researchers and practitioners characterize the nature of new work in the inspection field. In a sense, this structure also helps define a common vocabulary that depicts the software inspection area. Third, the survey presents an overview of the current state of research as well as an analysis of today's knowledge in the software inspection field.
We have to state that each survey has its limitations because it can only be a snapshot of the work that is currently in progress. Furthermore, a survey usually represents only a fraction of articles that are available on a subject. However, in this case we analyzed more than four hundred references. We are therefore convinced that this survey represents a good snapshot of the inspection-related work.

Acknowledgment

I am grateful to Jean-Marc DeBaud who participated in an earlier version of this survey as well as the anonymous reviewers for their comments on this chapter.

References

[1] Ackerman, A. F., Buchwald, L. S., and Lewsky, F. H., 1989. Software Inspections: An Effective Verification Process. IEEE Software, 6(3): 31-36.

[2] Ardissone, M. P. , Spolverini, M., and Valentini, M., 1998. Statistical Decision Support Method for In-process Inspections, Proceedings of the 4th International Conference on Achieving Quality In Software, pp. 135-143.

[3] Association of Computing Machinery, 1998. The ACM Digital Library. http://www.acm.org/dl/.

[4] Barnard, J. and Price, A., 1994. Managing Code Inspection Information. IEEE Software, 11(2):59-69.

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

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

[7] Basili, V. R. and Selby, R. W., 1987. Comparing the effectiveness of software testing techniques. IEEE Transactions on Software Engineering, 13(12):1278-1296.

[8] Belli, F. and Crisan, R., 1996. Towards Automation of Checklist-based Code-Reviews. Proceedings of the 8th International Symposium on Software Reliability Engineering.

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

[10] Blakely, F. W. and Boles, M. E., 1991. A Case Study of Code Inspections. Hewlett-Packard Journal, 42(4):58-63.

[11] Blalock, H. M., 1979. Theory Construction. Prentice Hall, Englewood Cliffs.

[12] Boehm, B. W., 1981. Software Engineering Economics. Advances in Computing Science and Technology. Prentice Hall.

[13] Bourgeois, K. V., 1996. Process Insights from a Large-Scale Software Inspections Data Analysis. Cross Talk, The Journal of Defense Software Engineering, 17-23.

[14] Briand, L., El Emam, K., Freimut, B., and Laitenberger, O., 1997. Quantitative Evaluation of Capture-Recapture Models to Control Software Inspections. Proceedings of the 9th International Symposium on Software Reliability Engineering.

[15] Briand, L. El Emam,, K., Freimut, B., Laitenberger, O., 2000. A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content, IEEE Transactions on Software Engineering, Vol. 26, No. 6.

[16] Briand, L., E -Emam, K., Fussbroich, T., and Laitenberger, O., 1998. Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects. Proceedings of the 20th International Conference on Software Engineering, pages 340-349.

[17] Briand, L. C., Differding, C. M., and Rombach, H. D., 1996. Practical guidelines for measurement-based process improvement. Software Process, 2(4):253-280.

[18] Briand, L. C., Freimut, B.G., Klein, B., Laitenberger, O., and Ruhe, G., 1998. Quality Assurance Technologies for the EURO Conversion - Industrial Experience at Allianz Life Assurance, in 2nd International Software Quality Week Europe, Brussels, Belgium.

[19] Bröhl, A.-P. and Dröschel, W., 1995. Das V-Modell. Oldenbourg.

[20] Brothers, L., Sembugamoorthy, V., and Muller, M., 1990. ICICLE: Groupware for Code Inspection. Proceedings of the ACM Conference on Computer Supported Cooperative Work, pages 169-181.

[21] Brykczynski, B. and Wheeler, D. A., 1993. An Annotated Bibliography on Software Inspections. ACM SIGSOFT Software Engineering Notes, 18(1):81-88.

[22] Cai, K., 1998, On Estimating the Number of Defects Remaining in Software, Journal of Systems and Software, vol. 40, pp. 93-114.

[23] Cheng, B. and Jeffrey, R., 1996. Comparing Inspection Strategies for Software Requirements Specifications. Proceedings of the 1996 Australian Software Engineering Conference, pages 203-211.

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

[25] Christenson, D. A., Steel, H. T., and Lamperez, A. J., 1990. Statistical quality control applied to code inspections. IEEE Journal Selected Areas in Communication, 8(2):196-200.

[26] Collofello, J. S. and Woodfield, S. N., 1989. Evaluating the effectiveness of reliability-assurance techniques. Journal of Systems and Software, 9:191-195.

[27] Conradi, R., Marjara, A.S., and Skatevik, B., 1999. Empirical Studies of Inspection and Test Data, in Proceedings of the First Conference on Product-Focused Process Improvement, Oulo, Finland.

[28] Cooper, H. M., 1982. Scientific guidelines for conducting integrative research reviews. Review of Educational Research, 52(2):291-302.

[29] Crossman, T. D., 1991. A Method of Controlling Quality of Applications Software. South African Computer Journal, 5:70-74.

[30] D'Astous, P., Robillard, P. N., 2000. Characterizing Implicit Information during Peer Review Meetings, Proceedings of the 22nd International Conference on Software Engineering, Limerick.

[31] Deck, M., 1994. Cleanroom Software Engineering to reduce Software Cost. Technical report, Cleanroom Software Engineering Associates, 6894 Flagstaff Rd. Boulder, CO 80302.

[32] DeMarco, T., 1982. Controlling Software Projects. Yourdon Press, N.Y.

[33] Dennis, A. and Valacich, J., 1993. Computer brainstorms: More heads are better than one. Journal of Applied Social Psychology, 78(4):531-537.

[34] Doolan, E. P., 1992. Experience with Fagan's Inspection Method. Software-Practice and Experience, 22(3):173-182.

[35] Dunsmore, A., Roper, M., Wood, M., 2000. Object-Oriented Inspection in the Face of Delocalisation, Proceedings of the 22nd International Conference on Software Engineering, Limerick.

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

[37] Dyer, M., 1992b. Verification-based Inspection. Proceedings of the 26th Annual Hawaii International Conference on System Sciences, pages 418-427.

[38] Eick, S. G., Loader, C. R., Long, M. D., Votta, L. G., and VanderWiel, S., 1992. Estimating Software Fault Content before Coding. Proceedings of the 14th International Conference on Software Engineering, pages 59-65.

[39] El Emam, K., Laitenberger, O., Harbich, H., 2000. The Application of Subjective Effectiveness to Controlling Software Inspections, Journal of Systems and Software, vol. 54, no. 2.

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

[41] Fagan, M. E., 1986. Advances in Software Inspections. IEEE Transactions on Software Engineering, 12(7):744-751.

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

[43] Franz, L. A. and Shih, J. C., 1994. Estimating the Value of Inspections and Early Testing for Software Projects. CS-TR- 6, Hewlett-Packard Journal.

[44] Fraunhofer Institute for Experimental Software Engineering, 1998. An Inspection Bibliography. http://www.iese.fhg.de/Inspections.

[45] Freedman, D. P. and Weinberg, G. M., 1990. Handbook of Walkthroughs, Inspections, and Technical Reviews. Dorset House Publishing, New York, 3rd edition.

[46] Gately, A.A., 1999, Design and Code Inspection Metrics, in International Conference on Software Management and Applications of Software Measurement, San Jose, Ca.

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

[48] Gintell, J., Houde, M., and McKenney, R., 1995. Lessons learned by building and using Scrutiny, a collaborative software inspection system. Proceedings of the 7th International Workshop on Computer-Aided Software Engineering, pages 350-357.

[49] Graden, M. E., Horsley, P. S., and Pingel, T. C., 1986. The Effects of Software Inspections on a major Telecommunications-project. AT&T Technical Journal, 65(3):32-40.

[50] Grady, R. B., 1994. Successfully applying software metrics. IEEE Computer, 27(9):18-25.

[51] Grady, R. B. and van Slack, T., 1994. Key Lessons in Achieving Widespread Inspection Use. IEEE Software, 11(4):46-57.

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

[53] Hetzel, W. C., 1976. An Experimental Analysis of Program Verification Methods. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science.

[54] Humphrey, W. H., 1995. A Discipline for Software Engineering. Addison-Wesley.

[55] International Software Engineering Research Network, 1998. Bibliography of the International Software Engineering Research Network. http://www.iese.fhg.de/ISERN/pub/isern_biblio_tech.html.

[56] Iisakka, J., Tervonen, I., 1998. Painless inprovements to the review process, Software Quality Journal, vol. 7, pp. 11-20.

[57] Jackson, A. and Hoffman, D., 1994. Inspecting module interface specifications. Software Testing, Verification and Reliability, 4(2):101-117.

[58] Jalote, P. and Haragopal, M., 1998. Overcoming the NAH Syndrome for Inspection Deployment. Proceedings of the Twentieth International Conference on Software Engineering, pages 371-378.

[59] Johnson, P., 1998a. The WWW Formal Technical Review Archive. http://zero.ics.hawaii.edu/johnson/FTR.

[60] Johnson, P. M., 1998b. Reengineering Inspection. Communications of the ACM, 41(2):49-52.

[61] Johnson, P. M. and Tjahjono, D., 1993. Improving Software Quality through Computer Supported Collaborative Review. Proceedings of the 19th International Conference on Software Engineering, pages 61-76.

[62] Johnson, P. M. and Tjahjono, D., 1997. Assessing software review meetings: A controlled experimental study using CSRS. ACM Press, pages 118-127.

[63] Johnson, p.M., Tjahjono, D., 1998, Does Every Inspection Really Need a Meeting, Journal of Empirical Software Engineering, vol. 3, no. 1, pp. 9-35.

[64] Jones, C., 1994. Gaps in the Object-oriented Paradigm. IEEE Computer, 27(6):90-91.

[65] Jones, C., 1996. Software Defect-Removal Efficiency. IEEE Computer, 29(4):94-95.

[66] Kamsties, E. and Lott, C. M., 1995. An empirical evaluation of three defect-detection techniques. In: Schäfer, W. and Botella, P., editors, Proceedings of the 5th European Software Engineering Conference, pages 362-383. Lecture Notes in Computer Science Nr. 989, Springer-Verlag.

[67] Kan, S. H., 1995. Metrics and Models in Software Quality Engineering. Addison-Wesley Publishing Company.

[68] Kaner, C. , 1998. The Performance of the N-Fold Requirement Inspection Method, Requirements Engineering Journal, vol. 2, no. 2, pp. 114-116.

[69] Kelly, J. C., Sherif, J. S., and Hops, J., 1992. An Analysis of Defect Densities found during Software Inspections. Journal of Systems and Software, 17:111-117.

[70] Kim, L. P. W., Sauer, C., and Jeffery, R., 1995. A framework for software development technical reviews. Software Quality and Productivity: Theory, Practice, Education and Training.

[71] Kitchenham, B., Kitchenham, A., and Fellows, J., 1986. The effects of inspections on software quality and productivity. Technical Report 1, ICL Technical Journal.

[72] Knight, J. C. and Myers, E. A., 1991. Phased Inspections and their Implementation. ACM SIGSOFT Software Engineering Notes, 16(3):29-35.

[73] Knight, J. C. and Myers, E. A., 1993. An Improved Inspection Technique. Communications of the ACM, 36(11):51-61.

[74] Kusumoto, S., 1993. Quantitative Evaluation of Software Reviews and Testing Processes. PhD thesis, Faculty of the Engineering Science of Osake University.

[75] Kusumoto, S., Chimura, A., Kikuno, T., Matsumoto, K., Mohri, Y., 1998. A Promising Approach to Two-Person Software Review in an Educational Environment, Journal of Systems and Software, no. 40, pp. 115-123.

[76] Laitenberger, O. and DeBaud, J.-M., 1997. Perspective-based Reading of Code Documents at Robert Bosch GmbH. Information and Software Technology, 39:781-791.

[77] Laitenberger, O., 2000. Cost-Effective Detection of Software Defects with Perspective-based Inspection, PhD-Thesis, University of Kaiserslautern, ISBN 3-8167-5583-6.

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

[79] Letovsky, S., Pinto, J., Lampert, R., and Soloway, E., 1987. A Cognitive Analysis of a Code Inspection. In Empirical Studies of Programming, pages 231-247.

[80] Levine, J. M. and Moreland, R. L., 1990. Progress in Small Group Research. Annual Review of Psychology, 41:585-634.

[81] Linger, R. C., Mills, H. D., and Witt, B. I., 1979. Structured Programming: Theory and Practice. Addison-Wesley Publishing Company.

[82] Macdonald, F., 1997. Assist v1.1 User Manual. Technical Report RR-96-199 [EFoCS-22-96], Empirical Foundations of Computer Science, (EFoCS), University of Strathclyde, UK.

[83] Macdonald, F. and Miller, J., 1995. Modelling Software Inspection Methods for the Application of Tool Support. Technical Report RR-95-196 [EFoCS-16-95], Empirical Foundations of Computer Science, (EFoCS), University of Strathclyde, UK.

[84] Macdonald, F., Miller, J., Brooks, A., Roper, M., and Wood, M., 1996b. Applying Inspection to Object-oriented Software. Software Testing, Verification, and Reliability, 6:61-82.

[85] Macdonald, F., Miller, J., Brooks, A., Roper, M., and Wood, M., 1996a. Automating the Software Inspection Process. Automated Software Engineering, 3(193):193-218.

[86] MacLeod, J. M., 1993. Implementing and Sustaining a Software Inspection Program in an R&D Environment. Hewlett-Packard Journal.

[87] Madachy, R., Little, L., and Fan, S., 1993. Analysis of a successful Inspection Program. Procceding of the 18th Annual NASA Software Eng. Laboratory Workshop, pages 176-198.

[88] Marciniak, J. J., 1994. Reviews and Audits. In: Marciniak, J. J., editor, Encyclopedia of Software Engineering, volume 2, pages 1084-1090. John Wiley and Sons.

[89] Martin, J. and Tsai, W.T., 1990. N-fold Inspection: A Requirements Analysis Technique. Communications of the ACM, 33(2):225-232.

[90] Mashayekhi, V., Drake, J. M., Tsai, W.T., and Riedl, J., 1993. Distributed, Collaborative Software Inspection. IEEE Software, 10:66-75.

[91] McCabe, T. J., 1976. A Complexity Measure. IEEE Transactions on Software Engineering, 2(4):308-320.

[92] McGibbon, T., 1996. A Business Case for Software Process Improvement. Technical Report F30602-92-C-0158, Data & Analysis Center for Software (DACS). URL: http://www.dacs.com/techs/roi.soar/soar.html.

[93] Miller, J., Wood, M., and Roper, M., 1998, Further Experiences with Scenarios and Checklists, Journal of Empirical Software Engineering, vol. 3, no. 3, pp. 37-64.

[94] Murphy, P. and Miller, J., 1997. A Process for Asynchronous Software Inspection. Proceedings of The 8th International Workshop on Software Technology and Engineering Practice, pages 96-104.

[95] Myers, G. J., 1978. A controlled experiment in program testing and code walkthroughs/inspections. Communications of the ACM, 21(9):760-768.

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

[97] OCLC, 1998. Online Computer Library Center. http://www.oclc.org/oclc/menu/home1.html.

[98] Parnas, D. L., 1987. Active Design Reviews: Principles and Practice. Journal of Systems and Software, 7:259-265.

[99] Parnas, D. L. and Weiss, D., 1985. Active Design Reviews: Principles and Practices. Proceedings of the 8th International Conference on Software Engineering, pages 132-136. Also Available as NRL Report 8927, 18 November 1985.

[100] Pedhazur, E. J., 1982. Multiple Regression in Behavioral Research. Hartcourt Brace College Publishers, second edition.

[101] Perpich, J., Perry, D., Porter, A., Votta, L., and Wade, M., 1997. Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development. Proceedings of the 19th International Conference on Software Engineering, pages 14-21.

[102] Perry, D. E., Porter, A., Votta, L. G., and Wade, M. W., 1996. Evaluating Workflow and Process Automation in Wide-Area Software Development. In: Montangero, C., editor, Proceedings of the 5th European Workshop on Software Process Technology, Lecture Notes in Computer Science Nr. 1149, pages 188-193, Berlin, Heidelberg. Springer-Verlag.

[103] Porter, A. A. and Johnson, P. M., 1997. Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies. IEEE Transactions on Software Engineering, 23(3):129-144.

[104] Porter, A. A., Siy, H., Mockus, A., and Votta, L., 1998. Understanding the Sources of Variation in Software Inspections. ACM Transactions on Software Engineering and Methodology, 7(1):41-79.

[105] Porter, A.A., Votta, L., Comparing Detection Methods for Software Requirements Inspection: A Replication using Professional Subjects, 1998. Journal of Empirical Software Engineering, vol. 3, no. 4, pp. 355-378.

[106] Porter, A. A., Siy, H., and Votta, L. G., 1995a. A Review of Software Inspections. Technical Report CS-TR-3552, UMIACS-TR-95-104, Department of Computer Science, University of Maryland, College Park, Maryland 20742.

[107] Porter, A. A., Siy, H. P., Toman, C. A., and Votta, L. G., 1997. An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development. IEEE Transactions on Software Engineering, 23(6):329-346.

[108] Porter, A. A. and Votta, L. G., 1997. What Makes Inspections Work? IEEE Software, pages 99-102.

[109] Porter, A. A., Votta, L. G., and Basili, V. R., 1995b. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. IEEE Transactions on Software Engineering, 21(6):563-575.

[110] Raz, T. and Yaung, A. T., 1997. Factors affecting design inspection effectiveness in software development. Information and Software Technology, 39:297-305.

[111] Reeve, J. T., 1991. Applying the Fagan Inspection Technique. Quality Forum, 17(1):40-47.

[112] Remus, H., 1984. Integrated Software Validation in the View of Inspections/Reviews. Software Validation, pages 57-65.

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

[114] Rosenthal, R., 1979. The "file drawer problem" and tolerance for null results. Psychological Bulletin, 86(3):638-641.

[115] Runeson, P., Wohlin, C., 1998. Journal of Empirical Software Engineering, An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections, vol. 3, no. 4, pp. 381-406.

[116] Russell, G. W., 1991. Experience with Inspection in Ultralarge-Scale Developments. IEEE Software, 8(1):25-31.

[117] Sandahl, K., Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M., and Ohlsson, N., 1998. An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspection, vol. 3, no. 4, pp. 327-354.

[118] Sauer, C., Jeffery, R., Lau, L., and Yetton, P., 2000. The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research, IEEE Transactions on Software Engineering, vol. 26, no. 1.

[119] Schneider, G. M., Martin, J., and Tsai, W. T., 1992. An experimental study of fault detection in user requirements documents. ACM Transactions on Software Engineering and Methodology, 1(2):188-204.

[120] Seaman, C. B. and Basili, V. R., 1998. Communication and Organization: An Empirical Study of Discussion in Inspection Meetings. IEEE Transactions on Software Engineering, 24(6):559-572.

[121] Shaw, M. E., 1976. Group Dynamics: The Psychology of Small Group Behaviour. McGraw Hill Inc.

[122] Shirey, G. C., 1992. How Inspections Fail. Proceedings of the 9th International Conference on Testing Computer Software, pages 151-159.

[123] Stein, M., Riedl, J., Harner, S., and Mashayekhi, V., 1997. A Case Study of Distributed, Asynchronous Software Inspection. Proceedings of the 19th International Conference on Software Engineering, pages 107-117. IEEE Computer Society Press.

[124] Strauss, S. H. and Ebenau, R. G., 1993. Software Inspection Process. McGraw Hill Systems Design & Implementation Series.

[125] Svendsen, F. N., 1992. Experience with inspection in the maintenance of software. Proceedings of the 2nd European Conference on Software Quality Assurance.

[126] Tervonen, I., 1996. Support for Quality-Based Design and Inspection. IEEE Software, 13(1):44-54.

[127] Tjahjono, D., 1996. Exploring the effectiveness of formal technical review factor with CSRS, a collaborative software review system. PhD thesis, Department of Information and Computer Science, University of Hawaii.

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

[129] Tripp, L. L., Stuck, W. F., and Pflug, B. K., 1991. The Application of Multiple Team Inspections on a Safety-Critical Software Standard. Proceedings of the 4th Software Engineering Standards Application Workshop, pages 106-111. IEEE Computer Society Press.

[130] Votta, L. G., 1993. Does Every Inspection Need a Meeting? ACM Software Eng. Notes, 18(5):107-114.
[131] Weinberg, G. M. and Freedman, D. P., 1984. Reviews, Walkthroughs, and Inspections. IEEE Transactions on Software Engineering, 12(1):68-72.
[132] Weller, E. F., 1992. Experiences with Inspections at Bull HN Information System. Proceedings of the 4th Annual Software Quality Workshop.
[133] Weller, E. F., 1993. Lessons from Three Years of Inspection Data. IEEE Software, 10(5):38-45.
[134] Wenneson, G., 1985. Quality Assurance Software Inspections at NASA Ames: Metrics for Feedback and Modification. Proceedings of the 10th Annual Software Engineering Workshop.
[135] Wheeler, D. A., Brykczinski, B., and Meeson, R. N., 1996. Software Inspection - An Industrial Best Practice. IEEE Computer Society Press.
[136] Wheeler, D. A., Brykczynski, B., and Jr., R. N. M., 1997. Software Peer Reviews. In: Thayer, R. H., editor, Software Engineering Project Management. IEEE Computer Society.
[137] Wiel, S. A. V. and Votta, L. G., 1993. Assessing Software Designs Using Capture-Recapture Methods. IEEE Transactions on Software Engineering, 19(11):1045-1054.
[138] Wohlin, C. and Runeson, P., 1998. Defect Content Estimations from Review Data. Proceedings of the 20th International Conference on Software Engineering, pages 400-409.
[139] Wood, M., Roper, M., Brooks, A., and Miller, J., 1997. Comparing and Combining Software Defect Detection Techniques: A Replicated Empirical Study. Proceeding of the 6th European Software Engineering Conference, Lecture Notes in Computer Science No 1301, ed. Mehdi Jazayeri, Helmut Schauer, pages 262-277.
[140] Yourdon, E., 1989. Structured Walkthroughs. Prentice Hall, 4th edition, N.Y.
[141] Yourdon, E., 1997. Death March Projects. Prentice Hall.