Fraunhofer Institut Experimentelles Software Engineering

A Measurement Framework for Software Inspections in the Quasar Context

Author:
Bernd Freimut

IESE-Report No.118.03/E
Version 1.0
September 1, 2003

A publication by Fraunhofer IESE

Abstract

Software inspections are accepted as a formal, effective, and efficient method for defect detection that is applicable in all development phases. In order to control the performance of software inspection processes the collection and analysis of data is recommended in the literature. Suggestions have been made for measuring selected aspects of inspections. Yet, we found no comprehensive survey of measuring software inspections.

In this preport, we present a five-level measurement framework that includes most applications of measurement in the context of software inspections. Quantitative models published in the literature are collected and discussed according to the framework's five level structure. This collection of models is intended to be a comprehensive survey of the topic. Moreover, the framework can be applied in future projects to derive a set of models that directly serve the intended measurement purpose.

Table of Contents

1 Introduction

Defect removal by verification and validation is a crucial point during software development, especially in the domain of embedded software developmend such as in the automotive industry. Cost, schedule and quality are directly addressed - three factors that are determinant for the success of a software development project. Defects must be found to ensure the desired product quality and they must be found in time and with acceptable effort to keep the project within its schedule and budget. This is stressed by Capers Jones [Jon97] who wrote *"the largest single identifiable cost element has been that of finding and fixing bugs"*. Stephen H. Kan [Kan95] mentioned *"defect removal is one of the top expense elements in any software project and also affects project scheduling. Effective defect removal can lead to reductions in the development cycle time and good product quality."* The basic idea is that the longer a defect remains in a software system the more effort must be spent on its repair. Consequently, a verification and validation program must include verification steps throughout the development process in order to avoid passing defects into the next life cycle phase.

A successful approach to verification that can be universally applied to the outcome of development activities is *software inspection*. Following a formally defined process, a peer group examines a software artifact for possible defects ensuring that the artifact is correct and conforms to product specifications and requirements. The purpose of software inspections is the detection and subsequent correction of defects.

In the course of the Quasar project inspections are tackled from two combinig viewpoints. The first viewpoint is that of how to perfom software inspections within the context of the Quasar approach. This entails especially how to use reading techniques in order to find defects in documents that are structured according to the Quasar approach. The second viewpoint, which is addressed by this report, is that of how to quantify inspections that are performed in general and in the Quasar context

The IEEE definition of software engineering demands three attributes of software development approaches: systematic, disciplined, and quantifiable. Software inspections being part of a development approach fulfil these requirements. The software inspection process is formally defined, its goals and requirements are stated and the roles for the participants are described (cf. Chapter 2). Besides, the necessity and importance of data collection is underlined in the literature. Ackermann et al. [ABL89] wrote *"the collection and analysis of data lies at the heart of the inspection process"*. Though there is a general agreement in the literature on this point and experience reports on the use of software inspections usually include an enumeration of collected data items, no systematic approach to the quantification of results from software inspections is presented.

Addressing this problem, the report has two goals: The first goal is to develop, starting with the work conducted in [Fus97], a framework for measuring software inspections following the idea of goal-oriented measurement. This framework should include the applications of quantified models in the context of software inspections, derive attributes that need to be measured from the applications, and finally propose operationally defined measures for these attributes.

The second goal is to identify and motivate activities that can contribute to the measurement of inspections that are performed within the Quasar context.

This report is structured as follows. Chapter 2 introduces software inspections. Chapter 3 presents the measurement framework. Chapter 4 discusses the applicability of the framework's models in industrial settings. Chapter 5 concludes the paper.

2 Software Inspections

This chapter provides a short introduction of software inspections. Although we assume that the reader familiar with the Quasar context is also familiar with software inspections, we nevertheless present the basic information on inspections here. The rationale for this is to make the report as self-containable as possible.

The basic ideas on software inspections are discussed in Section 2.1. The software inspection process consists of a series of activities which are described in detail in Section 2.2. The inspection team is more than a number of technical personnel taking part in the inspection. Several well-defined roles describe the respective part each participant plays during the different activities. These roles are presented in Section 2.3.

2.1 Basics of Software Inspections

Software inspections are one approach to the verification of software products, other approaches are reviews, walk-throughs, audits and mathematical proofs. These differ from each other in a number of factors, e.g. the method and scope of examination, their purpose, the people that are involved, and the way how they are involved.

The IEEE Standard for Software Reviews and Audits [IEE89] defines the objective of software inspections as follows:

"The objective of a software inspection is to detect and identify software element defects[1]. This is a rigorous, formal peer examination that does the following:

(a) *Verifies that the software element(s) satisfy its specifications*
(b) *Verifies that the software element(s) conform to applicable standards*
(c) *Identifies deviation from standards and specifications*
(d) *Collects software engineering data (for example, defect and effort data)*
(e) *Does not examine alternatives or stylistic issues"*

[1] The term *error* is commonly used with several possible definitions. The IEEE Standard Glossary of Software Engineering Technology [IEE90] gives these definitions:
"(a) An incorrect result. For example, a computed result of 12 when the correct result is 10.
(b) An incorrect step, process, or data definition. For example, an incorrect instruction in a computer program.
(c) A human action that produces an incorrect result. For example, an incorrect action on the part of a programmer or operator."
We assign definition (a) to the word *failure*, definition (b) to the word *fault*, and definition (c) to the word *error*. In this thesis, *defect* is used synonymously with *fault* as defined above.

Following Ebenau and Strauss [ES94], *"inspections are a means of verifying intellectual products by manually examining the developing product, a piece at a time, by small groups of peers to ensure that it is correct and conforms to product specifications and requirements."* Their purpose is *"the detection (and subsequent correction) of defects."* The basic concept behind inspections is *"that a small group of peers, concentrating on one part of the product, can detect more defects than the same number of people working alone"*.

The goals of software inspections are twofold. Inspections focus on the detection and subsequent correction of defects in software products, but the analysis of defect data can also be helpful for continuous improvement of the software development process. Furthermore, there are two requirements for software inspections: process modelling and measurement. The former is needed to identify software products to be inspected. Exit criteria of development activities determine when products require an inspection. The latter is necessary to follow the goal of product and process improvement. A complete survey on measuring software inspections will be presented in Chapter 3.

Since Fagan's first publication [Fag76] on software inspections, numerous variations have been proposed. The inspection process presented in the following sections does not follow a specific approach but is a generic process trying to capture the common aspects. Initially, inspections were used to examine design and code documents. In the meantime, they moved upstream in the development life cycle when the fact was recognized that the benefits of removing defects are highest if defects are detected in early phases. Today, any kind of software artefact can be subject of an inspection.

2.2 Inspection Activities

In detail, the software inspection process is as a sequence of six activities each having its own objectives. The following description of these activities is generic in the sense that they can be implemented in different ways for a special inspection process.

1. Planning
During planning, the objective is organization. The author prepares the material for the inspection. The organizer checks whether the materials meet the inspection entry criteria. The organizer is also responsible for the selection of the inspection team and the assignment of the roles to its members. He schedules the subsequent inspection activities and distributes the necessary material.

2. Overview (optional)
If an overview meeting is scheduled, its objective is education. The author gives a presentation of the material to be inspected that is attended by all inspectors.

The presentation introduces the inspectors to the material's function and relationships as well as to the techniques and representations that are used.

3. Detection
The objective is identification and documentation of defects. The inspectors use a reading technique to detect any defect in the distributed material. This activity can be implemented as individual preparation (inspectors get thoroughly familiar with the material on their own) or as an inspection meeting.

4. Collection
The objective is collection of and decision about defects. Potential defects found during detection are evaluated and if approved as defects they are added to the inspection defect list. This can be performed in an inspection meeting or by a single person via deposition meetings with each inspector or correspondence.

5. Correction
The objective is the correction of all defects. The author edits the material and deals with each defect on the inspection defect list.

6. Follow-up (optional)
The objective is the evaluation of the corrections made by author. After all defects have been resolved and the inspection's exit criteria are met, the moderator verifies the defect resolution.

2.3 The Inspection Team

Software inspections are conducted by a group of peers, typically comprising 3 to 5 people in a small team and up to 8 people in large teams. A number of specific roles are assigned to the participants having clear and specific responsibilities. Whether a role needs to be assigned to a team member may depend on the implementation of the inspection activities. The roles and their responsibilities are:

Organizer
The organizer plans all inspection activities.

Moderator
The moderator ensures that the inspection procedures are followed and other team members perform their responsibilities for each step. He moderates the inspection meeting if there is one. In this case, he is the key person in a successful inspection as he manages the inspection team and must offer leadership. Special training for this role is required.

Inspector
Usually all team members are assumed to be inspectors, regardless of their special role. Inspectors are responsible for detecting defects in the target software artefact.

Reader
At the inspection meeting, the reader leads 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 is responsible for the correction of defects during rework. During an inspection meeting, he 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 the inspection defect list during the inspection meeting.

Collector
The collector collects the defects found by the inspectors if there is no inspection meeting.

3 Measurement Framework

This chapter investigates the measurement of software inspections in more detail. Section 3.1 justifies why measurement provides for valuable information about software development processes. Section 3.2 discusses general aspects of measuring inspection processes. A sample proposal for data collection during inspections is presented and its insufficiencies are analysed. Section 3.3 presents a measurement framework that tries to structure measurement, its goals and components on different levels. Sections 3.4 and 3.5 contain definitions for basic attributes and models that are needed for the more complex models presented in Section 3.6. In this section, the proposed framework is applied to give an overview over measurement of inspection processes in the literature. A number of bibliographical references is analysed and their contents is presented according to the framework's structure. The findings of this overview are finally analysed in Section 3.7.

3.1 Why Do You Need Measurement?

Imagine a software quality assurance manager who wants to introduce software inspections into a development project. After carefully selecting the inspection process and the people to participate, he provides them with training on software inspections if necessary. He assigns them to the appropriate roles for a first inspection (e.g., of a requirements document), schedules the inspection activities, and distributes the necessary documents. Some days or weeks later, the organizer reports that the activities have been performed according to their description. Thus, the inspection process is completed.

Now what do the participants know about the success of the conducted software inspection? Do they know how good they performed? Do they have a precise understanding of what is "good"? Do they know how many defects were detected and, maybe more important, how many defects might not have been detected? Do they know which costs were associated with the inspection? Do they know if the inspection outperformed other verification and validation techniques in terms of effectiveness and efficiency?

By simply following instructions for software inspections one is able to detect and remove defects for sure, but one cannot provide for detailed information in order to answer questions like the ones stated above.

As a key concept to getting quantitative results in software engineering, measurement was integrated into the discipline's set of methods. Basili et al.

[BCR94b] performed a complete survey of measurement where the importance of measurement for software development is emphasized:

"Lord Kelvin's statement that "one does not understand what one cannot measure" is at least as true for software engineering as it is for any other engineering discipline. Measurement has been recognized as an indispensible prerequisite to introducing engineering discipline to the development, maintenance and use of software products."

It is not only about engineering discipline during different stages of the software life cycle, but measurement can also support different purposes, as pointed out by Briand et al. [BDR96]:

"Measurement is introduced in software organizations to gain quantitative insight into the development processes and the developed products. This is important in order to understand better the development process, to identify problems and improvement opportunities."

For these reasons, collecting additional information about the development process through a measurement program is useful and should be considered by project managers.

In general, we do know that inspections are a valuable part of a software development process. There is a lot of evidence that supports this, cf. the experience reports compiled by Wheeler et al. [WBM96]. However, inspections when implemented have a large amount of variation in their effectiveness and efficiency. With measurement one is able to optimize inspection performance - both maximizing the effect gained and minimizing the effort spent.

3.2 Basics of Measuring Software Inspections

The importance of applying measurement to software inspections is stressed in most of the related publications, e.g., expressed by Gilb in a humorous manner [Gil88]:

"The statistics principle: Inspection without statistics is like night driving without headlights; you may not see obstacles or opportunities until it is too late."

A lot of questions about software inspections can be answered by measurement. A few examples were given in the introduction to this chapter. In order to find answers to these questions, it is necessary to determine the purpose of measurement and decide which data needs to be collected. As an example, we have a look at the measurement program for software inspections that is presented by Ebenau and Strauss [ES94]. In general, they describe the possibilities and requirements of measuring software inspections to be:

"Inspection data analysis provides the ability to measure and control the performance of the inspection process, the quality of the emerging product, and the effectiveness of the development process. Accomplishing this depends on the collection and analysis of data about the identity of the work products that are inspected, the performance of the inspections, and the defects that are found."

In detail, they propose to collect the following data:

"During an inspection three types of data are recorded as an integral part of the process: project identification data, inspection performance data, and product defect data. This data provides a profile of the work product inspections, and is composed of

- Project identification data
- Project name (and any other relevant organizational information)
- Work product name (this may also include items such as release, account, etc.)
- Work product status (new, tested, modified, etc.)
- Moderator
- Meeting type (inspection, reinspection, overview)
- Inspection type (e.g., requirements, code, test plan)
- Inspection performance data
- Size of the work product (e.g., lines, pages)
- Date of the meeting
- Preparation time (hours)
- Examination time (hours)
- Number of inspectors
- Estimated rework (hours)
- Actual rework (hours)
- Rework completion date
- Work product disposition (e.g., accept, conditionally accept, rein-spect)
- Product defect data: Defect data classifies the problems identified in the product by the inspection meeting. (...) This data is composed of
    - Location of the defect (e.g., line #)
    - Description of the defect
    - Defect type (documentation, interface, logic, etc.)
    - Defect class (wrong, missing, extra)
    - Severity (major, minor)

When this information is collected and stored for each inspection, the result is a base of cumulative knowledge about the project, the product as a whole, each product component, and each inspection type."

Though a lot of data items are recommended to be collected, sole data collection should not be seen as sufficient. To extract valuable information from the collected data, it needs to be analysed. Yet, the proposal of Ebenau and Strauss lacks guidelines on how to perform this analysis. Besides, the purpose of data collection and the results that can be derived from the data are not clearly defined. Lacking this information their set of measures has to face the critique expressed by Basili et al. [BCR94b] with reference to the early days of software development:

"During the sixties, seventies and early eighties software measurement was in its primitive stage. This is reflected by the fact that measurement goals were neither explicit nor comprehensive. Implicit goals reflected the "production" view that there was a common set of goals shared by the entire software community. People disagreed over the usefulness of measures and measurements without realizing that they had different goals in mind. How else could it be possible that entire papers were devoted to discuss the usefulness of individual measures out of context."

Therefore, the framework that is to be developed in the following section has to mirror the aspect of purpose and goal of measurement, thus providing a context for the interpretation of collected data. As the focus of this thesis is on the performance of software inspections, measurement is concentrated on those aspects that Ebenau and Strauss listed as inspection performance data and product defect data. The organizational context provided by project identification data is important to identify sets of data that belong to a particular inspection, but is not subject of discussion here.

3.3 A Framework for Measuring Inspection Processes

This section describes a framework that is developed as a top-down approach for measuring software inspection processes. The rationale for the structure and components of the framework is basically the experience with measurement. Our framework can be interpreted as an application of the Goal/Question/Metric paradigm [Bas92, BCR94a]. However, the framework is tailored to the specific topic of measuring software inspections in contrast to the generic approach of GQM. Thus, we develop a framework in order to fit existing literature on measuring inspections into it.

The framework consists of 5 levels: application contexts, process characteristics, models, attributes, and measures. These levels are intended to provide guidance in the choice of measurement if they are used like a decision tree. By taking a sound decision on each level starting with the application context, it is possible to derive a set of models, attributes, and measures that address directly the intended purpose of measurement.

At the highest level, the proposed framework identifies groups of measurement applications that are used for a common purpose. Measurement applications for individual aspects of inspection processes are described on the second level whereas models, attributes, and measures that are essential to get quantitative results are covered on the lower levels. In detail, these levels are defined in the following way:

3.3.1 Application contexts

[Mer95] explains context as "the interrelated conditions in which something exists or occurs". At the framework's first level, an application context groups together a number of process characteristics. These are interrelated in the sense that the gained information is used for a specific purpose which the characteristics have in common. The measurement of inspection processes can be divided into the following three application contexts:

(a) Assessment of inspections
Several arguments qualitatively confirm that software inspections are an important activity during software development (cf. Section 1.1). However, the effects of conducting inspections in a software development project should be evaluated on a quantitative basis. Quantitative results on their costs and benefits and their impact on product quality and project schedule are needed. For example, it may be interesting to compare inspections to different testing strategies.

(b) Guidance for inspections
While conducting software inspections, guidelines for planning and monitoring them are necessary. The description of software inspections in Chapter 2 only includes the activities to be performed and the responsibilities of the people involved but, for example, no instructions at which rate a document should be inspected. Information is needed on how to integrate inspections into the project schedule, how many people should participate in the inspection, which qualifications they should have, how to perform inspection activities, which size an inspected document should have, and if inspections are performed in conformance to the guidelines. Besides, it must be decided if a reinspection of a document is necessary after the defects have been corrected.

(c) Improvement of inspection processes
After inspections have been implemented in an organization and its development projects, experience of conducting inspections is aggregated and improvement possibilities can be identified in order to maximize the benefits of inspections. Therefore, continuous improvement of the inspection process is essential, e.g., the guidelines for the inspection process have to be observed and, if necessary, corrected. Several other parameters can be varied in order to improve inspection performance like the number of inspection sessions (during detection), the number of participants, and the reading technique.

3.3.2 Process Characteristics

A process characteristic describes a single property of the inspection process that is evaluated by measurement. It delivers a special type of information that is needed in one of the described application contexts.

(a) Effectiveness
Effectiveness describes the ability of software inspections to detect defects.

(b) Efficiency
Efficiency characterizes the cost-effectiveness of detecting defects by inspections.

(c) Document quality
Software inspections help to improve the quality of the inspected document. After completion of the inspection process, the resulting document should have a certain quality level, e.g., with respect to the density of remaining defects.

(d) Impact on project schedule
Inspections need to be scheduled during a development project. The inspection activities consume additional effort and increase development time. Scheduling a meeting may even cause delays during a project.

(e) Status of inspection process
During a development project, the status of the inspections (i.e., the portion of documents that have already been inspected) may be of interest.

(f) Implementation of inspection activities
The generic inspection process as described in Chapter 2 offers the possibility of adapting the implementation of inspection activities to the project environment, e.g., conducting an inspection meeting or conducting a meeting-less inspection.

(g) Reading technique
Inspectors try to detect defects by using a specific technique, e.g., ad hoc, checklists, or perspective-based reading.

(h) Participants
While planning an inspection, the number of participants and their recommended qualification have to be decided.

(i) Guidelines
The participants of an inspection should follow guidelines for their work, e.g., with regard to the effort they spent on detection or the rate at which the document is inspected.

(j) Stopping criterion
A stopping criterion defines precisely when the detection activity should be ended.

(k) Reinspection
At completion of an inspection it is necessary to decide whether the inspected document needs a reinspection, e.g., because numerous changes have been made in order to correct the detected defects.

3.3.3 Models

A suitable explanation for model in [Mer95] is "a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs". Focusing on the mathematical aspects, a model is defined by Bender [Ben91] to be "an abstract, simplified, mathematical construct related to a part of reality and created for a particular purpose".

In accord with these definitions Lehman [Leh94] states that apart from mathematics "the term model, while widely applied, is generally less clear and rarely formally defined. Nevertheless the underlying concept holds good. A model reflects the properties of a theory and, thereby of some reality of which that theory is an abstract description. Loosely speaking one says that a model reflects some aspect or view of reality in the domain of discourse. Any number of such models can be created. Each will have been constructed by careful choice of axioms or assumptions to represent a particular viewpoint".

In this sense, a model is a mathematical description for a process characteristic that is based on certain assumptions and designed to reflect some aspect of reality in a particular environment. For any model, these assumptions have to be defined carefully as a model may only be used when its assumptions hold.

Following Briand et al. [BDR96], three different kinds of models are necessary to be developed for different process characteristics: Descriptive models are used to study characteristics of the software artifact or process that is examined. If a particular attribute needs to be evaluated and one of several alternative decisions has to be taken, an evaluation model is needed. As process characteristics may include prediction purposes, prediction models are the third category of models. The output of all three kinds of models is computed based on input variables, which we refer to as attributes.

3.3.4 Attributes

An attribute captures an abstract property of a software engineering artifact or a software process that is used in a model on the next higher level of the framework. Attributes do not have an operational definition as for example the attribute "document size" may have different meanings for a requirements document and a code document. The model's assumptions determine how to operationally derive a value for the attribute by selecting the appropriate measure.

3.3.5 Measures

Eventually, measures determine the value of attributes in an operational way. This can be derived easily from the following definition by Basili et al. [BCR94b]: "A software measure is a mapping from a set of objects in the software engineering world to a set of objects in the mathematical world. Objects in the software engineering world may be projects, products and processes. Objects in the mathematical world may be numbers or vectors of numbers. These mappings can be defined on different scales such as nominal, ordinal, interval, or ratio."

The framework is intended to be a means for structuring existing literature on measuring inspection processes. As application contexts and types are concepts that can be discussed without referring to a bibliographical reference, levels 1 and 2 of the framework are only described in detail in this section. The framework's lower levels will be covered in Section 3.6.

Process characteristics can be used in combination with more than one application context. For example, the characteristic effectiveness can be used within application contexts assessment, guidance, and improvement. Therefore, the framework need not eventually have a tree-like structure. Table 3.1 shows the possible combinations of application contexts and characteristics. Entries for the table cells will be made in Section 3.7. There, Table 3.2 fits the models that are discussed in Section 3.6 into the scheme given by Table 3.1.

	Assessment of inspections	**Guidance for inspections**	**Improvement of inspection processes**
Effectiveness			
Efficiency			
Document quality			
Impact on project schedule			
Status of inspection process		not applicable	not applicable
Implementation of inspection activities			
Reading technique			
Participants			
Guidelines			
Stopping criterion			
Reinspection			

Table 3.1: Application contexts and process characteristics

3.4 Basic Attributes

This section presents a set of basic attributes of inspection processes. The literature review in Section 3.6 takes into account more complex models which are composed using the basic attributes presented in this section. Thus, the definitions given here serve as a common baseline in order to present the information in the next sections in a consistent form. Otherwise, misinterpretations due to different definitions or notations could be possible. The set of basic attributes includes effort spent on inspections, duration of an inspection, number of defects, size and complexity of inspected products, and experience of the participants.

For the definitions, two conventions are used:

1. Some of the attributes that are defined in this section will be indexed, e.g., identifying a single inspection. When an attribute captures an aggregate value, e.g. the summation of an attribute of all inspections conducted, "+" replaces the index over which the summation is performed.

2. Sets are displayed in bold, for other variables the default font is used.

As the total number of inspections is needed for the definition of several basic attributes, we define it first. Let:

Definition 3.1

$i = 1 \ldots k$, where k is thetotal number of inspections conducted

3.4.1 Effort

When evaluating inspections, it is necessary to have a precise understanding of the effort consumed by the inspection process. Therefore, the effort spent on single inspection activities and on the process as a whole is to be recorded. The focus of this thesis is on measuring software inspections after they have been introduced into a software development organization. Hence, we do not consider the effort for this introduction, e.g., for management meetings or training.

Let:

Definition 3.2

$r = 1 \ldots q$, where q is the total number of participants of the inspection

We define the effort that a participant *r* spends in an inspection *i* on a single activity *a* as:

Definition 3.3

$\varepsilon_{i,a,r}$ = effort spent on activity *a* by participant *r* during inspection *i* [person hours], where:

Definition 3.4

$$a \in \{\text{Planning}, \text{Overview}, \text{Detection}, \text{Collecton}, \text{Correction}, \text{Follow-up}\}$$

The effort spent on inspection activity *a* by the whole inspection team is then defined as:

Equation 3.1

$$\varepsilon_{i,a,+} = \sum_{a} \varepsilon_{i,a,r}$$

In analogy, the effort spent on a single inspection *i* as a whole is defined as:

Equation 3.2

$$\varepsilon_{i,+,+} = \sum_{a} \varepsilon_{i,a,+}$$

3.4.2 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 that all people involved can really participate and fulfil their role. Thus, the interval for the completion of all activities will range from at least a few days up to a few of weeks. During this period, other work that relies on the artifact to be inspected is delayed and in extreme cases, developers may spend their time waiting for the inspection to be completed. If time to market is a critical issue during development, the effects of inspections on the development interval should be considered when discussing efficiency of inspections. For example, Votta discusses the effects of time loss due to meeting scheduling delay caused by people not being available at the same time [Vot93]. He advises to substitute inspection meetings by other forms of defect collection. According to his findings, meetings are not as beneficial as managers and developers think they are.

The duration of an inspection is defined as the interval between the availability of the material for inspection and the completion of the inspection process:

Definition 3.5

d = interval beteen the availability of material and completion of inspection [days]

In terms of inspection activities, this can be expressed as:

Equation 3.3

d = end date of follow-up – start date of planning [days]

3.4.3 Defects

The major goal of inspections is defect detection, collection, and correction. During detection, inspectors read a document with the goal of locating defects. During collection, the individual results of the detection activity are combined into a list of unique defects found during the detection step. During correction, the author removes the defects from the document following this list.

Optionally, for analysing the defects in detail, they may be classified according to a finite number of defect types. It is assumed that the defect types are mutual exclusive, i.e., that a defect has a unique defect type and that different inspectors consistently classify a defect as having the same defect type. Thus, an orthogonal defect classification scheme is assumed.

Defect data is central to evaluating effectiveness and efficiency of inspections as well as being necessary to control product quality. In the following, definitions for the sets of defects existing in the document prior to inspection and found during inspection with respect to inspection activity, inspector, and defect type are given.

As not all participants r necessarily take part in defect detection, we define the number of inspectors who conduct defect detection as:

Definition 3.6

$j = 1 \ldots l$, where l is the total number of inspectors

With respect to defect classification we define:

Definition 3.7

$t = 1 \ldots u$, where u is the number of defect types used

The set of all defects that exist in a document is in general neither known before nor after inspecting the document. Usually, only a subset of all defects will be detected by an inspection. However, the set of all unique defects is needed for evaluation purposes. Consequently, we define the set of unique defects of a special type *t* existing in a document prior to inspection *i* as:

Definition 3.8

$$\alpha_{i,t} = \{x \mid \text{x is a defect type that exists in the inspected doucment in inspection } i\}$$

The set of unique defects of all defect types existing in a document prior to inspection is defined as:

Equation 3.4

$$\alpha_{i,+} = \bigcup \alpha_{i,t}$$

The cardinality of this set can be expressed as:

Equation 3.5

$\left|\alpha_{i,+}\right| =$ total number of defects that exist in the inspected document prior to inspection *i*

It has to be considered that these defects may have different sources. Let us assume a waterfall process model consisting of requirements definition, design, coding, and testing. For example, if a defect is detected while inspecting a code document, this defect may have been introduced during coding, but it may have slipped through a previous design inspection. Thus, the origin of a defect does not need to be the phase where it is detected.

During detection, a subset of the total number of defects will be detected. We define the set of unique defects of a special type *t* found by inspector *j* during detection in inspection *i* as:

Definition 3.9

$$\beta_{i,j,t} = \{x \mid \text{x is defect of type t found by inspector j during detection in inspection i}\}$$

The set of unique defects of all defect types found by inspector *j* during detection in inspection *i* is defined as:

Equation 3.6

$$\beta_{i,j,+} = \bigcup_{t} \beta_{i,j,t}$$

The set of unique defects of a special type *t* found by all inspectors during detection in inspection *i* is defined as:

Equation 3.7

$$\beta_{i,+,t} = \bigcup_{j} \beta_{i,j,t}$$

The set of unique defects of all defect types found by all inspectors during detection in inspection *i* is defined as:

Equation 3.8

$$\beta_{i,+,+} = \bigcup_{j} \beta_{i,j,+}$$

The cardinality of this set can be expressed as:

Equation 3.9

$\left|\beta_{i,+,+}\right| =$ number of unique defects found during detection in inspection i

Following detection, collection is the next inspection activity to be performed. We define the set of unique defects of a special defect type *t* logged during collection in inspection *i* as:

Definition 3.10

$$\lambda_{i,j} = \{x \mid \text{x is a defect of type t logged during collection in inspection i}\}$$

and, as before, the set of unique defects of all defect types logged during collection in inspection *i* as:

Equation 3.10

$$\lambda_{i,+} = \bigcup_{t} \lambda_{i,t}$$

The cardinality of this set can be expressed as:

Equation 3.11

$\left|\lambda_{i,+}\right| =$ number of unique defects of all types logged during collection in inspection i

3.4.4 Size

Size is an important factor in order to characterize the document being inspected. It can be useful while determining how much effort should be spent on inspecting a document or if a document needs to be divided into several parts for inspection purposes.

Different kinds of documents will be produced during software development, e.g., requirements documents, design documents, test plans, and code. Due to the different structures of these document types various size measures have been proposed. Some of them consider more physical aspects of the document, e.g., the number of pages of a document or the number of lines of code, whereas other measures address logical aspects, e.g., the number of modules a system is composed of during design or the number of function points.

A general size measure that is applicable to any kind of document is the number of pages:

Definition 3.11

$s_{physical}$ = number of pages in a document

It has to be considered that the amount of information on a page can vary tremendously between documents of different types and even documents of the same type.

Especially for code documents, “lines of code (LOC)” is proposed most often as a size measure (e.g., [Hum95]). This measure needs to be defined carefully in a particular environment because it can be a source of controversies, which lines to count (e.g., empty lines, commentary lines, or multi-statement lines).

Definition 3.12

s_{code} -= number of lines of code (LOC)

As there is no LOC count e.g., for a design document at the time of inspection, an analysis of inspections using LOC as a consistent size measure must be postponed. Therefore, some authors propose to use an estimated LOC count based on historical data [Hum95].

An example of a size measure that is based on logical aspects of a document is the function point measure [Jon97]: “A function point is a synthetic metric that is comprised of the weighted totals of the inputs, outputs, inquiries, logical files or user data groups, and interfaces belonging to an application.” The idea behind function points was to create a measure that deals with external features of the software that are important to business owners and users. Therefore, the measure is available early in the development life cycle and can be used for planning the development project. Besides, it stays constant independent of the programming language, design technology, or development skills involved [Alb94].

Definition 3.13

$s_{logical}$ = number of function points

3.4.5 Complexity

To detect defects inspectors must understand the artifact being inspected. Depending on its contents this task may be more or less demanding. This may influence the result of the inspection. For example, if a document is hard to understand, inspection effectiveness may decrease because defects remain undetected due to a lack of understanding.

When talking about the understandability of software, systems, and the related documents, the term complexity is often used though an exact definition is difficult as explained by Zuse [Zus94]:

"Complexity is both difficult to define precisely and to quantify. However, we can say that the complexity of an object is some measure of the mental effort required to understand that object."

It is worth mentioning that this definition refers to the psychological complexity of an artifact in contrast to other facets of complexity like computational complexity of software, which is a machine-oriented view of a program's algorithmic complexity. Thus, complexity measures must reflect what is complex for humans. Numerous complexity measures have been proposed, for a detailed discussion cf. [Zus94]. Jones [Jon97] gives a good overview on different forms of complexity. He describes 20 varieties of complexities and their respective measures if available.

3.4.6 Experience

When looking at a software development process, the experience of the participants is another issue of interest. Their knowledge and familiarity with the relevant methods and techniques, e.g., software inspections, are important factors determining effectiveness and efficiency of process enactment. However, experience is far more difficult to measure, as it has neither a physical representation nor a general definition. Thus, the kind of experience that is to be measured needs to be defined precisely in combination with an appropriate model. Often, these models are built on subjective ratings and ordinal scales of measurement are used.

For example, Basili [Bas92] presents a characterization of the experience of a team with regard to a particular process. After defining the steps of an education and training program, the experience is measured by a subjective rating per person as follows:

- 0 - none
- 1 - have read the manuals
- 2 - have had a training course
- 3 - have had experience in a laboratory environment
- 4 - have used on a project before
- 5 - have used on several projects before
- x - no response"

In another example, the report analyses a system test process. One of the aspects that are characterized is the familiarity of the system domain to the tester:

- 0 - domain new to me
- 1 - have had a course in the subject domain
- 2 - have built or tested one system in this domain
- 3 - have built and tested at least one system in this product line
- 4 - have built and tested several systems in this domain
- 5 - have tested and built several systems in this product line"

Analysing software inspections, the experience of the people participating in the inspection with

- the inspection process itself,
- the application domain of the product being developed, and
- the techniques used in the artifact being inspected

may be issues of interest.

3.5 The Basic Set of Descriptive Models

Building on the basic attributes defined in the previous section this section introduces descriptive models that will be used when discussing models in the literature. Models for the average cost of finding and fixing a defect during inspections and for other defect detection techniques are presented. In general, the average cost would be calculated by collecting the individual cost of finding and fixing a number of defects, summarizing these costs, and divide the sum by the number of defects. Since we collect data for costs at the level of inspection activities, this procedure is not applicable here. Therefore, the total costs for an inspection (or another defect detection technique) are normalized by the number of defects found in order to calculate the average cost for the scope of this thesis.

3.5.1 Average Cost of Finding and Fixing a Single Defect During Inspections

We assume that the cost of finding and fixing defects during inspection is equal to the effort spent on inspection because effort is the largest cost driver for conducting inspections. As explained earlier, we only consider this cost, not other expense factors that may be related to inspections, e.g., training or external consultants.

Using the definitions for effort (Equation 3.2) and defects (Equation 3.11), the average cost of finding and fixing a single defect during an inspection *i* is defined as:

Equation 3.12

$$\bar{\varepsilon}_{inspection,i} = \frac{\varepsilon_{i,+,+}}{\left|\lambda_{i,+}\right|}\left[\frac{\text{person-hours}}{\text{defect}}\right]$$

Generalizing this value over the inspections conducted within an organization, the average cost of finding and fixing a single defect during *k* inspections for an organization is defined as:

Equation 3.13

$$\bar{\varepsilon}_{inspection,+} = \frac{\sum_i \bar{\varepsilon}_{inspection,i}}{k}\left[\frac{\text{person-hours}}{\text{defect}}\right]$$

3.5.2 Average Cost of Finding and Fixing a Single Defect for Other Techniques

In general, it is a valid assumption that the software development process of an organization includes other defect detection techniques than software inspections. Usually, these will follow after inspections have been conducted and may include testing and maintenance of the product. For evaluating the benefits of inspections and comparing them with other defect detection techniques, it is necessary to define the sets of defects detected by other techniques and the effort spent for detecting these defects.

Let *f* be a defect detection phase (excluding software inspections) of a software development process. In analogy to the definitions concerning inspections, we define:

Definition 3.14

$$\alpha_{f,t} = \{x \mid \text{x is a defect of type t that exists in the examined document prior to phase f}\}$$

Thus, the set of unique defects of all defect types existing in the examined document prior to phase *f* is defined as:

Equation 3.14

$$\alpha_{f,+} = \bigcup_{t} \alpha_{f,t}$$

The set of defects of defect type *t* found during defect detection phase *f* is defined as:

Definition 3.15

$$\lambda_{f,t} = \{x \mid \text{x is a defectof type t found during defect detection phase f}\}$$

Thus, the set of all defects found during defect detection phase *f* is defined as:

Equation 3.15

$$\lambda_{f,+} = \bigcup_{t} \lambda_{f,t}$$

With ε_f representing the effort spent on finding and fixing these defects, the average cost of finding and fixing defects for a single defect detection phase is defined as:

Equation 3.16

$$\bar{\varepsilon}_f = \frac{\varepsilon_f}{\left|\lambda_{f,+}\right|} \left[\frac{\text{person-hours}}{\text{defect}}\right]$$

The average cost of finding and fixing a single defect using other defect detection techniques than software inspection for an organization is then defined as:

Equation 3.17

$$\bar{\varepsilon}_+ = \frac{\sum \bar{\varepsilon}_f}{\tilde{k}} \left[\frac{\text{person-hours}}{\text{defect}}\right]$$

where $\tilde{k}$ is number of defect detection phases analysed.

3.6 Literature Review

The framework presented in Section 3.3 is now used to give a structured overview of measurement of inspection processes in the literature. A number of bibliographical references will be analysed according to that framework. For each reference, a description of the proposed model is given in three parts: First, the model is characterized along the framework as complete as information is available. The model and its respective attributes and measures are organized according to the framework's levels 3, 4 and 5 and the mapping between adjacent levels is described. Where possible, the model is given using the terms originally used by the author and re-expressed using the definitions of this thesis. In addition, the process characteristc and context are determined. Second, the underlying assumptions of the model are defined. Third, a discussion concludes the description and may provide various pieses of information about the model, e.g., on the environment where the model was developed, the purpose it was designed for, and the experience made. Table 3.2 shows how the models can be used for the application contexts and characteristics presented in Table 3.1.

3.6.1 Descriptive Models

In this section, quantitative models are discussed that operationalize abstract attributes of inspection processes. We present models for the effectiveness of individual inspection activities, the effectiveness of the inspection process as a whole, and the efficiency of inspection processes.

3.6.1.1 Effectiveness of Individual Inspection Activities

Depending on the actual implementation of detection, collection, and correction during an inspection, the effects of the individual activities can be evaluated as all of them have an impact on the results of the inspection.

Meeting Loss and Meeting Gain

If the collection activity of the inspection process is implemented as a meeting, the collection of the defects is more than simply merging the individual findings of the inspectors. Synergy effects of working collaboratively in a meeting have to be considered [Vot93]. Defects that no inspector found during detection may be found during the meeting (so-called meeting gain). On the other hand, inspectors may not mention some of the defects they found during detection (so-called meeting loss). Based on the previous definitions we can define meeting gain and meeting loss for inspection *i*:

Definition 3.16

$$M_{Loss,i,+} = \left|\beta_{i,+,+} \setminus (\lambda_{i,+} \bigcap \beta_{i,+,+})\right|$$

Definition 3.17

$$M_{Gain,i,+} = \left|\lambda_{i,+} \setminus (\lambda_{i,+} \bigcap \beta_{i,+,+})\right|$$

and if we are only interested in defects of a certain type *t*, then:

Definition 3.18

$$M_{Loss,i,t} = \left|\beta_{i,+,t} \setminus (\lambda_{i,t} \bigcap \beta_{i,+,t})\right|$$

Definition 3.19

$$M_{Gain,i,t} = \left|\lambda_{i,t} \setminus (\lambda_{i,t} \bigcap \beta_{i,+,t})\right|$$

False Positives

After collecting the defects found by the inspectors, $\lambda_{I,+}$ is the set of unique defects that are assumed to be in the inspected document. As the inspectors are not as familiar with a document as the author is, and defects are just logged but not discussed at an inspection meeting, it is possible that $\lambda_{I,+}$ contains issues that are not real defects. The author will discover these issues during correction. Issues that are logged as defects but turn out not to be defects are called false positives. False positives cause unnecessary effort during correction because the author will spend some time on checking the logged issue before uncovering it as a false positive.

With Equations 3.5 and 3.11 we define the number of false positives as:

Definition 3.20

$$\text{number of false positives} = \left|\lambda_{\mathrm{i},+} - (\lambda_{i,+} \cap \alpha_{i,+})\right|$$

Bad fixes

During correction the author of the inspected document has to resolve all defects that were logged during collection. Each element of $\lambda_{I,+}$ that is no false positive forces a correction or fix in the document. This activity itself may be error-prone because it cannot be assumed in general that all fixes are correct, i.e., a correction may not resolve the detected defect or introduce other defects. A faulty correction is called bad fix. Thus, an inspection may remove fewer defects than detected. Bad fixes cause trouble because resulting defects may slip into the next development phase if the document is not reinspected after correction. However, for this thesis we do not consider the possibility of bad fixes as no evaluation of their impact is presented in the literature.

3.6.1.2 Effectiveness of an Individual Inspection

In order to characterize the effectiveness of an individiual inspection, i.e., the complete set of inspection activities, typically information about defect number and defect type is used.

Chillarege et al., "Orthogonal Defect Classification (ODC)" [CBC+92]

Application context

Assessment of inspections, Guidance of inspections

Process characteristic

Effectiveness

Model

$$\text{Defect Distribution} = \{|\alpha_{i,1}|, \ldots, |\alpha_{i,u}|\}$$

Attributes

(a) The number of defects for each defect type

Measures

(a) $|\alpha_{i,t}|$ for all t

Assumptions

In the application of their ODC model, Chillarege et al. use a specific defect classification model in order to characterize the defects found. Particularly the defect classification attribute "Trigger" is used for analyzing inspection defects. The rationale underlying the definition of these types is that each defect type represents the issue that the inspector was thinking about when detecting the defect. Each defect type is related to a specific aspect of experience that is required to detect that defect. Thus, an analysis of the defect distribution can reveal, what trigger detected many defects and what triggers detected only few defects. This information can help to estimate, whether the right kinds of defects have been detected or whether certain types of defects might still be undetected. For example, re-inspection decisions can be made interpreting this model.

Discussion

The model based on the defect classification attribute has been reported in a case study [CHBC93]. Here it is an open question to which extent the scheme can be adapted in a different context, or to which extent the rationale of the ODC scheme can be transferred into other contexts allowing for the same kind of analysis.

Ebenau and Strauss, Defect Density [ES94]

Application context

Assessment of inspections, Guidance of inspections

Process characteristic

Effectiveness

Model

$$\text{Detected Defect Density} = \frac{\text{Number of defects}}{\text{Size}} = \frac{|\lambda_{i,+}|}{s}$$

Attributes

(a) Number of detected defects
(b) Size of instected document

Measures

(a) $|\lambda_{i,+}|$
(b) s Size

Assumptions

When comparing detected defect densities of different processes it has to be taken into account that the initial defect density influences the detected defect density. Therefore, the model assumes that factors impacting this initial quality of the document such as language, complexity, domain, etc are constant when comparing different processes.

Discussion

This model is often used, as its measures can be determined after completion of the collection step.

3.6.1.3 Effectiveness of the Inspection Process as a Whole

When looking at effectiveness of an inspection as a whole, its ability to detect and remove defects is to be evaluated. The effects of individual inspection activities are only taken into account in an aggregated form. The following models present various approaches to effectiveness of inspections.

Fagan, "Design and code inspections to reduce errors in program development" [Fag76]

Application context

Assessment of inspections

Process characteristic

Effectiveness
(Though Fagan uses the term "error detection efficiency", effectiveness is more appropriate, as the aspect of cost is not included in his model.)

Model

$$M_f = \frac{\text{defects found by an inspection}}{\text{total defects in product before inspection}} \times 100$$
$$= \frac{|\lambda_{i,+}|}{|\alpha_{i,+}|} \times 100$$

Attributes

(a) Defects found in a product during inspection
(b) Total defects in the product before inspection

Measures

(a) $|\lambda_{i,+}|$
(b) $|\alpha_{i,+}|$

Assumptions

Fagan's model can only provide for precise values after completion of product development or even after extensive field usage as the total number of defects is not known at the time of inspection.

Discussion

Fagan introduces error detection efficiency in order to evaluate detailed-design inspections, code inspections, and unit test inspections.

Humphrey, "A Discipline for Software Engineering", Chapter 8 [Hum95]

Application context

Assessment of inspections

Process characteristic

Effectiveness

Model

$$\begin{aligned}\text{Yield} &= \frac{\text{number of defects removed during inspection}}{\text{total number of defects in document at the time of inspection}} \times 100 \\ &= \frac{|\lambda_{i,+}|}{|\alpha_{i,+}|} \times 100\end{aligned}$$

Attributes

(a) number of defects removed during inspection
(b) total number of defects in the inspected document at the time of inspection

Measures

(a) $|\lambda_{i,+}|$
(b) $|\alpha_{i,+}|$

Assumptions

Humphrey states that yield "cannot be precisely calculated until after the reviewed program has been thoroughly tested and extensively used. Even then, there could be some latent defects still to be discovered." The total number of defects known to be in a document may change after every defect detection phase as defects from all previous development phases can be detected. Therefore, it is assumed that it can be decided for a detected defect when it was injected. This information is necessary to calculate yield correctly.

However, Humphrey argues that useful early approximations can be made providing for upper limits of yield. These approximations could also take into account historical data, e.g., on the proportion of design defects that were generally missed in design inspections, but detected in later defect detection phases.

Discussion

Yield as proposed by Humphrey is similar to the effectiveness model proposed by Fagan. Additionally, Humphrey addresses the problem of calculation by discussing the effects of multiple defect detection phases and shows how the yield measure may change its value after each phase.

Kan, "Metrics and Models in Software Quality Engineering", Chapter 6 [Kan95]

Application context

Assessment of inspections

Process characteristic

Effectiveness

Model

Defect Removal Effectiveness (DRE)

$$DRE = \frac{\text{Defects removed (at end of development phase)}}{\text{Defects existing on phase entry} + \text{Defects injected during development}} \times 100\%$$

$$= \frac{|\lambda_{i,+}|}{|\alpha_{i,+}|} \times 100\%$$

Attributes

(a) defects removed by inspection at the end of the development phase
(b) defects existing on entry of development phase
(c) defects injected during development phase

Measures

(a)	$\lvert\lambda_{i,+}\rvert$
(b) + (c)	$\lvert\alpha_{i,+}\rvert$

Assumptions

Kan assumes a development process that is divided into different phases. For the development phases before testing, the development activities themselves are subject to defect injection, and the inspections at the end of the phase activities are the key vehicle for defect removal. For defects detected during inspections or testing, he emphasizes another possibility of injecting defects by faulty corrections (so-called bad fixes).

To derive an operational definition, defects need to be classified in terms of the development phase in which the defects are found (and removed) and the phases in which the defects are injected. Kan proposes the use of a matrix to trace this phase information. Thus, calculations of effectiveness measures are straightforward.

Discussion

Kan performs a literature review on effectiveness models discovering that these differ little from one to another. However, he thinks that the subtle differences could become significant if there are separate phases of development and inspections before code integration and testing. He argues that when the inspection of an early phase took place, the defects from later phases could not have been injected in the product yet (and therefore could not be detected by an inspection following the early phase). Thus, he proposes a phase-based model of effectiveness.

Briand et al., “Building Resource and Quality Management Models for Software Inspections” [BLW97]

Application context

Assessment of inspections

Process characteristic

Effectiveness

Model

$$\text{Effectiveness} = \frac{\text{Number of detected defects}}{\text{Size of inspected document}} = \frac{|\lambda_{i,+}|}{s}$$

Attributes

(a) Number of detected defects
(b) Size of inspected document

Measures

(a) $|\lambda_{i,+}|$
(b) s = number of operations specified

Assumptions

This model makes a major assumption: All inspected documents have a similar actual defect density. The actual defect density is the total number of defects of a document per unit size. The authors argue that they could make this assumption because all inspections they analysed were conducted in the same organization, within the same application domain, and during a short period of time.

Discussion

The authors look for an effectiveness model that does not use the total number of defects in a document, as this number is not known in general. Therefore, they decide to measure effectiveness as the density of defects detected. Thus, they are at least able to compare different inspections though depending on how close to reality the assumption above is.

3.6.1.4 Efficiency

Efficiency characterizes the cost-effectiveness of detecting defects by software inspections. The following models present various approaches to efficiency of inspections.

Collofello and Woodfield, “Evaluating the Effectiveness of Reliability-Assurance Techniques” [CW89]

Application context

Assessment of inspections

Process characteristic

Efficiency

Model

$$M_C = \frac{\text{Cost saved by the process}}{\text{Cost consumed by the process}}$$

Attributes

(a) Costs saved by the process
From their point of view, the costs saved by an inspection “would be the amount of resources (some combination of time, money, etc.) that would have been expended” to handle the defects that were removed due to inspection during development. It is stated that a good estimation for this attribute is needed. Therefore, the need for data collection is stressed.

(b) Costs consumed by the process
The costs consumed by an inspection “correspond to the resources spent in handling errors.” This includes their detection and removal.

Measures

(a) cf. Assumptions for the model
(b) $\varepsilon_{I,+,+}$

Assumptions

The model assumes that an organization has *n* reliability-assurance processes, 1,…*n* , that are applied sequentially. Based on this assumption, the costs saved by some reliability-process *v* is calculated as the sum of the costs associated with having to use processes *(v+1)* to *n* to handle the defects detected by process *v*. The costs associated with handling defects in a process *w*, where $v < w \leq n$, can be calculated as the average resources to detect and fix a defect in process *w* times the expected number of defects detected by process *w*. The average resources for detecting and fixing a defect in process *w* must be based on historical data. The expected number of defects detected by process *w* can be calculated as the defect-detectin effectiveness of the *w*th process times the number of defects from process *v* which still have not been detected. The defect-detection effectiveness of a process is also based on historical data with an assumption that the defect-detection effectiveness of process *n* is 1. The number of defects detectable by process *v* is calculated based on the error-detection effectiveness of the processes that precede process *w*.

Additionally, this calculation of the costs associated with handling the defects by following defect-detection processes assumes that each defect that is detected by process *v* is related to exactly one defect that would be detected by process *w*.

Discussion

Collofello and Woodfield extend previous work on inspection effectiveness. They are convinced that a reliability-assurance technique can have low defect-detection effectiveness but still be considered worthwhile because it can save money. Therefore, they develop ideas on "cost effectiveness".

The results of a case study show that code inspections were more effective at finding defects than design inspections. At the opposite, design inspections proved to be far more efficient than code inspections though these still showed a return on investment.

Kusumoto, "Quantitative Evaluation of Software Reviews and Testing Processes" [Kus93]

Application context

Assessment of inspections

Process characteristic

Efficiency

Model

$$M_k = \frac{\Delta C_t - C_r}{C_t + \Delta C_t} = \frac{|\lambda_{i,+}| \times \bar{\varepsilon}_{testing} - |\lambda_{i,+}| \times \bar{\varepsilon}_{inspection,+}}{|\alpha_{i,+}| \times \bar{\varepsilon}_{testing}}$$

The model depends on three attributes: C_r, the actual cost of inspections, C_t, the actual cost of testing, and ΔC_t, the reduction of the testing cost compared to the virtual testing cost. Kusumoto defines virtual testing cost as the cost of detecting and removing all defects in the test phase provided no inspection is executed. Using this term, he assumes that the testing cost is reduced by ΔC_t compared to the virtual testing cost because the defects that have been found by inspections generate no effort during testing.

"Intuitively, is a ratio of the reduction of the total costs to detect and remove all faults from the design documents and program code using design and code reviews in a project to the virtual testing cost (of the program code)."

Attributes

(a) Cost of inspections
The cost spent on inspections during a project. It is calculated as the product of the number of defects found and removed during design and code inspections and the average cost of detecting and removing a defect during inspections. (Measures: c, d, e)

(b) Reduction of testing cost compared to virtual testing cost
The cost that is not spent on testing because defects were removed during inspections. It is calculated as the product of the number of defects found and removed during inspections and the average cost of detecting and removing a defect during testing. (Measures: c, d, e)

(c) Virtual testing cost
The cost of detecting and removing all defects during test provided no inspection is performed during the project. It is calculated as the product of the number of defects introduced into design and code documents and the average cost of detecting and removing a defect during testing. (Measures: a, b, e)

Measures

(a) $|\alpha_{i,+}|$ where *i* a design inspection

(b) $|\alpha_{i,+}|$ where *i* a code inspection

(c) Number of defects detected and removed during design inspections: $|\lambda_{i,+}|$, where *i* is a design inspection

(d) Number of defects detected and removed during code inspections: $|\lambda_{i,+}|$, where *i* is a code inspection

(e) Average cost to detect and remove a defect during inspections: $\bar{\varepsilon}_{inspection,+}$

(f) Average cost to detect and remove a defect during testing: $\bar{\varepsilon}_{testing}$, where testing is the only other defect detection technique considered for the calculation of the average cost

Assumptions

Before introducing his efficiency model Kusumoto states his assumptions about the development process. A standard waterfall model consisting of six phases (concept exploration and feasibility analysis, requirement specification, design, implementation, testing, and maintenance) is simplified as follows: Starting from an error-free specification that is not changed during development it is assumed that defects are introduced in the subsequent design and implementation phases. Inspections are performed at the end of these phases and detect a number of defects that are removed. Finally, the product passes a test phase. Supposedly all defects from design and implementation are detected and removed during testing.

Discussion

In his dissertation, Kusumoto addresses the need to "evaluate and prove the effectiveness of software reviews with respect to software development cost" by proposing a new model for the quantitative analysis of inspections with respect to their efficiency (which he calls "cost effectiveness"). He criticizes that other models (like Collofello/Woodfield) do not take into account the total cost to detect and remove all defects from a product by inspections and testing. To address this task, he introduces the concept of virtual testing cost.

Grady and van Slack, "Key Lessons In Achieving Widespread Inspection Use" [GS94]

Application context

Assessment of inspections

Process characteristic

Efficiency

Model

$$ROI = \frac{\text{engineering time benefits}}{\text{engineeritng time costs}}$$

Attributes

(a) engineering time benefits
(b) engineering time costs

Measures

(a) cf. Assumptions for the model
(b) $\varepsilon_{i,+,+}$

Assumptions

In [Gra92], a number of detailed calculations is presented applying this model. Historical data from sample organizations is used to calculate the necessary parameters, e.g., average number of design defects, average number of defects found per inspection hour, ratio of cost to find and fix defects during test to cost during design. The costs for inspections include not only the cost of conducting inspections, but also training and start-up costs. It is pointed out that "potential added revenue from faster time to market improves the ROI even more."

Discussion

Grady and Van Slack present a sample cost-benefit analysis for design inspections yielding a ROI of 10.4. They argue that this value is a much better return than that of many other R&D investments. From their point of view, there are two additional advantages: They assume that little risk is involved in investing into inspections, and initial investments are said to pay off quickly. Besides, they mention that training and start-up costs occur only once per team.

The calculations concerning engineering time benefits and costs only include the time to detect defects, but do not consider the time to fix a defect. Therefore, the ROI model does not mirror that fixing a defect during testing may be more expensive than fixing it earlier in the development cycle.

Briand et al., "Building Resource and Quality Management Models for Software Inspections" [BLW97]

Application context

Assessment of inspections

Process characteristic

Efficiency

Model

$$\text{Efficiency} = \frac{\text{Number of detected defects}}{\text{Size of inspected document} \times \text{Preparation Effort}}$$
$$= \frac{\left|\lambda_{i,+}\right|}{s \times \varepsilon_{i,preparation,+}}$$

Attributes

(a) Number of detected defects
(b) Size of inspected document
(c) Preparation effort

Measures

(a) $|\lambda_{i,+}|$
(b) s = number of operations specified
(c) $\varepsilon_{i,preparation,+}$

Assumptions

The authors develop this model using the effectiveness model presented in Section 3.6.1.4. Therefore, the same major assumption is made: All inspected documents have a similar actual defect density. The actual defect density is the total number of defects of a document per unit size.

Discussion

The authors define efficiency by normalizing effectiveness by the amount of preparation effort spent. They argue that this would capture the detection cost-effectiveness, the “amount of effectiveness” achieved per unit of effort spent on defect detection. Effort spent on other inspection activities, e.g., correction, is not considered for this model.

3.6.2 Evaluation Models

Evaluation models are needed to evaluate a software inspection and compare its performance with other inspections or other defect detection techniques. We present models that can be used to evaluate the effectiveness and efficiency of an inspection.

3.6.2.1 Effectiveness

Ebenau and Strauss, “Statistical Process Control” [ES94]

Application context

Guidance for inspections, improvement of inspection processes

Process characteristic

Effectiveness

Model

$$\mathrm{LCL} \le \text{Defect Density} \le \mathrm{UCL}$$
$$\Leftrightarrow \mathrm{LCL} \le \frac{|\lambda_{i,+}|}{s} \le \mathrm{UCL} \quad \text{; where}$$

$$\mathrm{LCL} = \max(0, \overline{\mu} - 3 \times \sqrt{\frac{\overline{\mu}}{\max(s_i)}})$$

and

$$\mathrm{UCL} = \overline{\mu} + 3 \times \sqrt{\frac{\overline{\mu}}{\overline{s}}}$$

and

$$\overline{\mu} = \frac{\sum_{i=1}^{k} |\lambda_{i,+}|}{\sum_{i=1}^{k} s_i}$$

Attributes

(a) For a set of inspections: Size of each inspected document
(b) For a set of inspections: Number of detected defects

Measures

(a) s_i = Size of each inspected document
(b) $|\lambda_{i,+}|$

Assumptions

One major assumption of this model is that all documents have a similar initial defect density.

Discussion

This model assumes that the inspection process is stable and operates with a given performance indicated by its average detected defect density. Single inspections that deviate too much from this average performance are candidates for which the process might have failed and are therefore to be investigated. Operationally, upper and lower control limits (UCL and LCL) are determined. If the defect densisty from the inspection under investigation is above the upper or below the lower control limit, then this inspection has to be investigated more closely.

3.6.2.2 Efficiency

Briand et al., "Building Resource and Quality Management Models for Software Inspections" [BLW97]

Application context

Assessment of inspections, guidance for inspections, improvement of inspection processes

Process characteristic

Efficiency

Model

$$\begin{aligned}\text{Efficiency} &= \text{Size}^{a} \times \text{Preparation Effort}^{b} \times c \\ &= s^{a} \times \varepsilon_{i,preparation,+}{}^{b} \times c\end{aligned}$$

Attributes

(a) Size of inspected document
(b) Preparation effort

Measures

(a) s= number of operations specified
(b) $\varepsilon_{i,preparation,+}$

Assumptions

As for their descriptive model, the authors make the major assumption that all inspected documents have a similar actual defect density. The actual defect density is theb total number of defects of a document per unit size.

Discussion

Analysing inspection data from customer projects, the authors found that "efficiency decreases exponentially with both document size and preparation effort". Performing a regression analysis, they developed a linearized multivariate model based on standardized data. The constants *a*, *b*, and *c* for the exponential model above can be derived from the linear model.

For performing the evaluation of an inspection, the authors propose to calculate the actual efficiency value using their descriptive efficiency model (cf. Section 3.6.1.4) and the predicted efficiency value using this model. A scatterplot then shows predicted vs. actual efficiency. By locating the inspection on this scatterplot, it can be evaluated if the inspection performed like a typical inspection (within a given confidence interval). If located outside the confidence interval, the inspection performed more or less efficient than expected. Thus, changes to the inspection process, e.g., the introduction of a new reading technique, can be evaluated.

3.6.3 Prediction Models

A prediction model defines precisely a functional relationship between independent and dependent variables. Data on the independent variables is used to calculate a predicted value for the dependent variables. In this section, we present a prediction model for inspection efficiency.

3.6.3.1 Efficiency

Briand et al., "Building Resource and Quality Management Models for Software Inspections" [BLW97]

Application context

Guidance of inspections

Process characteristic

Efficiency

Model

$$\begin{aligned}\text{Efficiency} &= \text{Size}^{a} \times \text{Preparation Effort}^{b} \times c \\ &= s^{a} \times \varepsilon_{i,preparation,+}{}^{b} \times c\end{aligned}$$

Attributes

(a) Size of inspected document
(b) Preparation effort

Measures

(a) *s*= number of operations specified
(b) $\varepsilon_{i,preparation,+}$

Assumptions

As for their descriptive model, the authors make the major assumption that all inspected documents have a similar actual defect density. The actual defect density is the total number of defects of a document per unit size.

Discussion

The authors assume an increasing exponential relationship with decreasing slope between effectiveness and preparation effort. In addition, they assume a decreasing exponential relationship between effectiveness and the number of specification operations. Following the same procedures as for their efficiency evaluation model, the authors derive both the exponential model above and the values for the parameters.

The model is proposed to be used "for planning purposes, i.e., to predict the preparation effort before an inspection is conducted, or for control purposes, i.e., to achieve a higher level of document quality".

3.6.3.2 Effectiveness (Capture-Recapture Models)

In biology, capture-recapture models are used to estimate the size of animal populations. The basic idea of combining these models and inspections is that a capture-recapture model can be used to estimate the number of defects that remain in a document after inspection. By comparing the number of defects that were detected during inspection with the estimated number of total defects, the number of remaining defects can be estimated. Taking into account this number and the accuracy of the capture-recapture model used, it can be decided whether the inspection can be stopped or whether the document has to be reinspected. The overall objective of this approach is to maximize the proportion of defects that are found before releasing the document to the next phase. For a detailed discussion of capture-recapture models cf. [Fre97] and [BEFL00].

3.6.3.3 Effectiveness (MARS)

Briand et al. 2003, "Using Multiple Adaptive Regression Splines to Support Decision Making in Code Inspections", [BFF03]

Application context

Guidance of inspections

Process characteristic

Effectiveness

Model

$$\begin{aligned}\text{Defects} &= a_0 + a_1 \times BF1 + a_2 \times BF4 + a_3 \times BF9 + a_4 \times BF11\\ &+ a_5 \times BF14 + a_6 \times BF25 + a_7 \times BF27\\ &+ a_8 \times BF28 + a_9 \times BF30\end{aligned}$$, with

BF1 = max(0, EFFORT - 30.000);
BF4 = max(0, PARTICIP - 10.000) * BF1;
BF9 = max(0, DLOC - 3000.000) * BF1;
BF11 = max(0, SESSIONS - 4.000) * BF1;
BF14 = max(0, 0.250 - RATE) * BF1;
BF19 = max(0, EFFORT - 3600.000);
BF21 = max(0, EFFORT - 2100.000);
BF24 = max(0, 4200.000 - EFFORT);
BF25 = max(0, PARTICIP - 9.000) * BF24;
BF27 = max(0, LOC - 29.999) * BF19;
BF28 = max(0, DLOC - 4.000) * BF21;
BF30 = max(0, 0.433 - RATE) * BF21;

Attributes

(a) Size of inspected document in LOC (LOC)
(b) Size of inspected document in DLOC (DLOC)
(c) Effort (EFFORT)
(d) Number of participants (PARTICIP)
(e) Inspection rate (RATE)
(f) Number of Sessions (SESSIONS)

Measures

(a)+(b)	*s*= Size of docuument
(c)	$\varepsilon_{i,+,+}$
(d)	$\varepsilon_{i,+,+}$
(e)	q_{i}
(f)	$\frac{\varepsilon_{i,preparation,+}}{s_i}$
(g)	the number of meetings held

Assumptions

none

Discussion

The authors use data mining techniques in order to reveal relationships between the number of defects and other, independent variables, such as effort, size, and others. The relationships are expressed as linear splines, upon which a linear regression is performed.

The model is proposed to be used for planning purposes, i.e., to plan important parameters before an inspection is conducted, or for control purposes, i.e., to achieve a higher level of document quality.

3.6.3.4 Causal Factors for Effectiveness and Efficiency

This section gives an overview of hypotheses on driving factors for effectiveness and efficiency of software inspections. These hypotheses try to predict causal relationships between one or more independent variables (the driving factors) to one or more dependent variables (effectiveness and efficiency). In order to improve effectiveness and efficiency, it may be valuable to analyse these relationships and consider the important factors while implementing the inspection process. The factors are organized with respect to the dependent variable.

Effectiveness

- **Number of participants**
  Porter et al. [PSV95] argue that an inspection team that is composed of several reviewers allows a wide variety of defects to be found since each reviewer relies on different expertise and experiences when inspecting. The larger and more varied the team, the better the coverage of the inspected document. Thus, effectiveness should increase with increasing team size.

- **Number of sessions**
  Porter et al. [PSV95] identify the number of inspection sessions the artifact undergoes in the inspection process as another factor influencing effectiveness. In their opinion, multiple-session inspections, possibly with different teams of inspectors, will find more defects as long as some important or subtle defects escape detection by any one inspection session. They also propose that splitting one large team inspection into multiple sessions with smaller teams might be more effective. The effect of using multiple inspections led to the N-fold inspection method [SMT92].
  For multiple-session inspections, the way of conducting the sessions is another option. Sessions can be scheduled in parallel - with each session inspecting the same version of the artifact - or in sequence - with defects found in one session being repaired before going on to the next session. Porter et al. [PSV95] argue that parallel sessions will be more effective only if different teams find few defects in common. Sequential sessions may find more defects since cleaning out old defects might make it easier to find new ones.
  Porter, Votta, Siy, and Toman [PVST95] ran an experiment at AT&T on a project that is developing a compiler and environment to support developers of the AT&T 5ESS telephone switching system. The subjects were all of the team's six members plus five other developers. All were experienced, and all had received training on inspections within five years of the experiment. The project conducted more than 100 code inspections during which three independent variables were manipulated: the team size, the number of inspections sessions, and the coordination sessions. The results showed the following: There was no difference in effectiveness neither between small teams and large teams nor between 2-session inspections held in parallel and those held in sequence. 2-session, 2-reviewer inspections were more effective than 1-session, 4-reviewer inspections, but 2-session, 1-reviewer inspections were not more effective than 1-session, 2-reviewer inspections.
- **Collection technique**
  Porter et al. [PSV95] discuss the effects of using different collection techniques during the inspection process. A collection meeting may be held (group-centered) or not (individual-centered). Votta [Vot93] shows that meetings do not create as much synergy in finding defects as previously believed.
- **Qualification of participants**
  In their article, Porter et al. [PSV95] present a number of case studies and experiments that deal with the costs and benefits of inspections.
  In one of these case studies, Rifkin and Deimel suggest teaching program comprehension techniques during code inspection training classes in order to improve program understanding during preparation and inspection. Based on historical data, they argued that while inspections reduced the number of defects discovered by testing, they did not significantly

decrease the number of customer-identified defects. To test this hypothesis, they collected data from three software development groups, each composed of 30 to 35 professionals. One group was given 1.5 days training in program reading comprehension. The data showed that the number of customer-reported defects dropped by 90% after the inspectors received this training, while results of the two other groups of reviewers showed no change.

– **Inspection rate**
  Buck [Buc81] conducted a study at IBM to identify variables that would differentiate high quality inspections from low quality ones. His idea was that the number of defects found in an inspection is not an adequate indicator because it is influenced by the quality of the artifact being inspected. He collected data from 106 code inspections of a single piece of Cobol source code. The collected data showed that code inspections conducted at a rate of less than 125 NCSL (non-commentary source code lines) per hour found significantly more defects. There was no difference in defect detection capability between teams of 3, 4, and 5 inspectors. Effectiveness was also independent of major defects found per hour, but additional preparation resulted in more defects being found. Thus, the study suggests that quality inspections are a result of following a low inspection rate.
  Ebenau [Ebe94] applies the means of statistical process control to inspection data from a project that enhanced the features of a local telephone switching system. During the project, 25 detailed design and code inspections were conducted. Ebenau states three factors that are known to affect the detection of defects: examination rate, preparation rate, and work product size. Using charts, he shows that defect density of the documents inspected depends on the examination rate. Besides, he demonstrates that examination and preparation rate as well as preparation rate and work product size are directly related. Therefore, Ebenau recommends to conduct inspections in the analysed project at preparation and inspection rates of 150 lines per hour, inspecting less than 200 hundred lines of material at a time.
  Christenson et al. [CHL90] use statistical quality control, too. They applied them to code inspections during AT&T's 5ESS Switch project. The factors they found to correlate well with the density of defects detected were the amount of preparation effort, the inspection rate and the size of the unit of code. The authors found the density of discovered defects to have an inverse relationship to the inspection rate.
– **Preparation effort**
  Christenson et al. [CHL90] report on experience from inspections where the inspectors prepare individually for the inspection meeting by studying design and code documents. The authors found preparation effort to have high positive correlation with the density of defects found.

- **Size of inspected document**
  Christenson et al. [CHL90] also emphasize that the size of the code document being inspected was the major factor influencing the inspection rate and preparation effort. In their environment, larger units of code tended to receive proportionally less preparation and were inspected at a higher rate. This resulted in a lower density of discovered defects for larger units of code.
- **Defect detection technique**
  Porter et al. [PSV95] explain that defect detection techniques range in prescriptiveness from intuitive, nonsystematic procedures (such as ad hoc or checklist techniques) to explicit and highly systematic procedures (such as scenarios or correctness proofs). Other defect detection techniques include reading by stepwise abstraction, defect-based reading, and perspective-based reading. The choice of a defect detection technique may have an impact on effectiveness as well. For example, more systematic techniques may help inexperienced inspectors.
- **Tool support**
  Perry et al. [PPVW96] hypothesize that the effectiveness of any tool supporting the software inspection process depends on how well the tool co-exists with the process it supports. If it is congruent with the process, then they assume a chance that the process is enhanced. If not, the tool may be a severe hindrance. With respect to HyperCode, a collaborative inspection tool they developed, they suppose that HyperCode inspections will be no less effective than traditional inspections. The authors announce that they are conducting an experiment to evaluate this hypothesis.

Efficiency

Porter et al. [PSV95] show several factors that have an impact on the effort that is spent on inspecting a document.

- **Number of participants**
  Larger teams require more effort since more people analyse the artifact which may be unfamiliar to them. This also reduces the time they can spend on other development work. Besides, it is more difficult for everyone to contribute fully during the meeting because of limited air time.
- **Number of sessions**
  The inspection effort will increase as the number of inspection sessions conducted during the inspection process grows.
  Conducting multiple sessions in parallel will increase the effort slightly because the author needs to collect the reports and sort out which issues from different reports actually refer to the same defect in the artifact.
- **Collection technique**
  Using meetings as the technique for collecting the defects from the in-

spectors increases the effort compared with other collection techniques (cf. [Vot93]).

- **Post-collection meeting**
  Finally, Porter et al. [PSV95] discuss the use of post-collection feedback. Some authors argue that a brainstorming meeting should be held after the inspection meeting to determine the root cause of each issue recorded in the meeting. Doing so, the development team may learn why defects were made, and how they could have been avoided. However, the additional meeting will increase the effort.
- **Tool support**
  Perry et al. [PPVW96] argue that tool-based inspection will require no more human effort than traditional inspections, but will incur fewer indirect costs (e.g., travel, conference calls, photocopying).

3.7 Critique

Finally, this section analyses the results of the literature review. A general critique of the framework is performed. The effectiveness and efficiency models are discussed in more detail as the following chapters of this thesis concentrate on these models.

The Framework in General

The framework proved its usefulness as a means for fitting existing literature on measuring software inspections into it. Though models are not described consistently in different bibliographical references, the necessary information for each framework level can be extracted. The presentation of all models using the framework and the definitions from Sections 3.4 and 3.5 makes a comparison of strengths and weaknesses easier.

Table 3.2 gives an overview over all models included in Section 3.6 and shows how they relate to the two highest levels of the framework, application contexts and process characteristics. For two reasons, Table 3.2 is only sparsely filled. First, some process characteristics are discussed in the literature, but not formulated as quantitative models. Second, other process charateristics are of interest in our point of view, but are not discussed in the reviewed literature at all. It should be considered that the literature review could not be comprehensive though more literature was considered than listed in the bibliography. Besides, the framework may evolve over time as more process charatcersitics are identified because of more experience with measuring inspection processes.

	Assessment of inspections	Guidance for inspections	Improvement of inspection processes
Effectiveness	Fagan	Chillarege et al.	
	Humphrey	Ebenau and Strauss	
	Kan	Briand et al. 2003	
	Briand et al.		
Efficiency	Collofello/Woodfield	Briand et al. 1997	Briand et al., 1997
	Kusumoto		
	Grady/van Slack		
Document quality	Capture/Recapture	Chillarege et al.	
Impact on project schedule			
Status of inspection process		not applicable	not applicable
Implementation of inspection activities			
Reading technique			
Participants			
Guidelines			
Stopping criterion	Capture/Recapture	Ebenau and Strauss	
Reinspection	Capture/Recapture	Ebenaus and Stauss	
		Chillarege et al.	

Table 3.2: How models address application contexts and process characteristics

Before analysing the effectiveness and efficiency models in detail, we want to state some prerequisites for all quantitative models in the context oft software inspections:

- A model should be usable soon after completion of an inspection.
- Therefore, data collection for the necessary measures should be feasible.
- Where information is missing, the estimation of values should be explained.

Effectiveness Models

The effectiveness models can be separated into two groups: On the one hand the model of Briand et al., on the other hand the models of Fagan, Humphrey and Kan. The major difference between the model of Briand et al. and the other models is the assumption that all inspected documents have a similar actual defect density. This assumption must hold whenever the model of Briand et al. is to be used. Thus, it may be especially difficult to compare results from different environments with this model.

The models of Fagan, Humphrey, and Kan are similar to each other as they relate the number of defects detected to the total number of defects. These models face the problem that the total number of defects in a document is not known in general. Fagan's article is the oldest related publication. He uses the effectiveness model for the assessment of inspections after completion of the development project. Thus, the model cannot be used right after an inspection is completed. He does not discuss the usage of the model thoroughly, maybe due to the fact that the potential of measurement was not recognized in the mid 1970's.

Humphrey and Kan try to provide a solution to the problem of early availability. Humphrey explains how to calculate a preliminary yield value after completion of inspection. This value can be used as an upper limit for yield. The value may be adjusted when more defects are found in later defect detection phases that slipped through the original inspection of the document. Hence, the total number of (known) defects increases and yield decreases. However, the origin of a defect has to be identified for this purpose as explained by Kan. Therefore, a specific level of traceability between documents that are produced in different phases has to be established. For example, if a defect found during code inspection is diagnosed as being introduced during design, the respective design document must be identifiable in order to adjust its effectiveness.

As another way of determining effectiveness right after completion of the inspection process, Humphrey proposes to estimate the number of total defects using the number of defects detected as input. He advises the use of defect profiles based on historical data, e.g., a profile "that shows that generally one more defect is later found in integration test, system test, or customer use for every three defects found in compile and unit test". Though this procedure may work in the context of a personal software process where only small documents (e.g., the code for a module) are inspected, it is not clear if the procedure will scale up to development projects.

Efficiency Models

Similar to the effectiveness models, the descriptive models for assessing the efficiency of software inspections can be separated in two groups: On the one hand the model of Briand et al., on the other hand the models of Kusumoto, Grady/van Slack, and Collofello/Woodfield. Again, the major difference is the assumption of similar actual defect density for the model of Briand et al.

The other group of models relates in different ways the costs spent on the inspection process to the costs that are potentially saved. Hence, these models interpret efficiency as "cost-effectiveness", i.e., the effect that is achieved by conducting inspections. They do not simply calculate efficiency as cost per defect.

In order to determine the costs potentially saved by inspections all models assume that the defects detected by an inspection would be found in another defect detection phase if the inspection had not been conducted. The models of

Kusumoto and Grady/van Slack use testing as the only other defect detection technique and assume that all defects found during inspection would also be found during testing. In general, this assumption does not hold because testing is not perfect - defects may slip through into maintenance.

The most detailed description of how to calculate the costs potentially saved by inspections is given by Collofello and Woodfield. They simply assume a number of defect detection phases in general. Besides, they consider varying effectiveness levels for these phases, i.e., the probability that a defect would be detected may change from phase to phase. The need for historical data is emphasized for all three models. Besides, all models assume that a defect detected by inspection will result in exactly one defect in a later defect detection phase. This is questionable as, for example, a design defect that is passed into coding may cause several defects in the code. In addition, the models do not use defect classification. Distinguishing between defects of various defect types would make the efficiency models more precise as the variation of the costs for finding and fixing a defect in a phase may be captured better by grouping defects to defect types.

4 Data Analysis in the Framework's Context

The preceding framework synthesized the available quantitative models for measuring inspection processes. Thus, they discuss how to define operationally important properties of the inspection process. However, in order to apply these models successfully, appropriate data collection and analysis is necessary.

In this section we will discuss for selected models, what prerequisites in terms of data collection and analysis have to be available.

4.1 Desciptive Models

The descriptive models in Section 3.6.1 define, how to operationally measure aspects of the inspection process. In order to apply these models it is necessary to collect data to the required measures.

Here we identify two open problems: the collection of these data, which we discuss in Section 4.1.1, and the definition of the right measures in the context of the defect types, specifically the Orthogonal Defect Classification Scheme (cf. Section 3.6.1.2), which we discuss in Section 4.1.2.

4.1.1 Data Availability

The models of Fagan, Humphrey, Kan, Collofello/Woodfield, Kusumoto, and Grady/van Slack assume that a detailed data collection exists across the entire development life-cycle.

For example, the models of Fagan, Humphrey and Kan rely on the measure $|\alpha_{i,+}|$. In order to obtain this measure it is necessary to classify defects in defect detection activities that follow the inspection process with respect to their origin. The efficiency models of Collofello/Woodfield, Kusumoto, and Grady/van Slack require this data as well and moreover require effort data from both the inspection and testing process.

Such data, however, are often not collected in industral settings. Therefore one has to take the decision, whether to use other, maybe less appropriate models, or wheter missing data has to be obtained from other sources.

Briand et al. [BFF00] proposed an approach, in which a combination of project data and expert opinion is used in order to apply these models. (Originally, their approach was applied using the Kusumoto model. However, the approach can also be used to detemine the other models as well.)

4.1.2 ODC Construction

As indicated in Section 3.6.1.2, the ODC model is an interesting idea to measure the effectiveness of an inspection or set of inspections. However, it assumes that an appropriate set of defect types (i.e. defect classification scheme) is used. The defect types used in the contexts reported in the literature depend strongly on the reading techniques used during the inspection process and the domain of the inspected product.

Therefore it is necessary to adapt the set of defect types if other reading techniques and products are to be used. However, it is an open question, how this adaptation can be performed and whether similar interpretations can me made.

5 Conclusion

This report presented a measurement framework of software inspections that contained decisions to be taken in the inspection process and quantitative models that support these decisions.

One benefit of this report is therefore to act as a reference document, of how various characteristics and attributes of the inspection process can be measured. In addition, this report identifies areas in this framework where quantitative support is still lacking or where it is difficult to readily implement the proposed models in industrial settings, such as the automotive domain in the context of the Quasar context.

Therfore, we conclude that, in order to contribute to the Quasar Project, it is useful to tackle the problem of applying the identified models in industry. Two problems have been identified in this context: the availability of data collection in industrial settings, and the availability of appropriate defect typologies for the ODC-Model. While the first problem has been addressed by some other work, the second one is seen as fuitful to explore in the Quasar context. For an elaboration of this problem, the interested reader might want ot refer to [FD03].

6 References

[ABL89] A. Frank Ackerman, Lynne S. Buchwald, and Frank H. Lewski. Software Inspections: An Effective Verification Process. IEEE Software, 6(3):31–36, May 1989.

[Alb94] Allan J. Albrecht. Function Points Analysis. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 518–524. John Wiley and Sons, Inc., New York, 1994.

[Bas92] Victor R. Basili. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical Report CS-TR-2956, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, College Park, MD 20742, September 1992.

[BCR94a] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley and Sons, Inc., New York, 1994.

[BCR94b] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Measurement. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 646–661. John Wiley and Sons, Inc., New York, 1994.

[BDR96] Lionel C. Briand, Christiane M. Differding, and H. Dieter Rombach. Practical Guidelines for Measurement-Based Process Improvement. Technical Report ISERN–96– 05, Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, D– 67661 Kaiserslautern, Germany, 1996.

[Ben91] Edward A. Bender. *An Introduction to Mathematical Modeling*. Krieger Publishing Company, Malabar, Florida, reprint edition, 1991.

[BLW97] Lionel C. Briand, Oliver Laitenberger, and Isabella Wieczorek. Building Resource and Quality Management Models for Software Inspections. Technical Report ISERN– 97–06, Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, D–67661 Kaiserslautern, Germany, 1997.

[BEFL00] Lionel C. Briand Khaled El-Emam, Bernd Freimut, and Oliver Laitenberger, A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content, IEEE Transactions on Software Engineering, vol26, no. 6, pp. 518-540, 2000.

[BFF00] Lionel C. Briand, Bernd Freimut, Ferdinand Vollei, Assessing the Cost-Effectiveness of Inspections by Combining Project Data and Expert Data, Proceedings of the 11th International Symposium on Software Reliability Engineering, pp.124-135, 2000.

[BFF02] Lionel Briand, Bernd Freimut, Ferdinand Vollei, Using Multiple Adaptive Regression Splines to Support Decision Making in Code Inspections, accepted for Publication in Journal of Systems and Software, 2003.

[Bou96] Karen V. Bourgeois. Process Insights from a Large-Scale Software Inspections Data Analysis. *Cross Talk, The Journal of Defense Software Engineering*, 9(10):17–23, October 1996.

[BP94] Jack Barnard and Art Price. Managing Code Inspection Information. *IEEE Software*, 11(2):59–69, March 1994.

[Buc81] F. O. Buck. Indicators of Quality Inspections. Technical Report TR21.802, IBM Corp., Kingston, NY, September 1981.

[CBC+92] Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray, and Man-Yuen Wong, Orthogonal defect classification -- A concept for in-process measurements, *IEEE Transactions on Software Engineering*, vol. 18, pp. 943--956, Nov. 1992.

[CHBC93] Jarir K. Chaar, Michael J. Halliday, Inderpal S. Bhandari, and Ram Chillarege, In-Process Evaluation for Software Inspection and Test, IEEE Transactions on Software Engineering, vol. 19, pp. 1055--1070, Nov. 1993.

[CHL90] Dennis A. Christenson, Steel T. Huang, and Alfred J. Lamperez. Statistical Quality Control Applied to Code Inspections. *IEEE Journal on Selected Areas in Communcations*, 8(2):196–200, February 1990.

[CW89] James S. Collofello and Scott N. Woodfield. Evaluating the Effectiveness of Reliability-Assurance Techniques. *Journal of Systems and Software*, 9(3):191–195, 1989.

[Ebe94] Robert G. Ebenau. Predictive Quality Control with Software Inspections. *Cross Talk, The Journal of Defense Software Engineering*, 7(6):9–16, June 1994.

[ES94] Robert G. Ebenau and Susan H. Strauss. *Software Inspection Process*. Systems Design & Implementation Series. McGraw-Hill, Inc., New York, 1994.

[Fag76] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.

[Fag86] Michael E. Fagan. Advances in Software Inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751, July 1986.

[Fow86] Priscilla J. Fowler. In-Process Inspections of Workproducts at AT&T. *AT&T Technical Journal*, 65(2):102–112, March/April 1986.

[Fre97] Bernd Freimut. Capture-Recapture Models to Estimate Software Fault Content. Master's thesis, University of Kaiserslautern, June 1997.

[FD03] Bernd Freimut, Christian Denger: A Defect Classification Scheme for the Inspection of QUASAR Requirement Documents, IESE-Report No. 076.03/E, 2003.

[Fus97] Thomas Fussbroich. Measuring Inspection Processes. Master's thesis, University of Kaiserslautern, 1997.

[GG93] Tom Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley Publishing Company, Wokingham, England, 1993.

[Gil88] Tom Gilb. *Principles of Software Engineering Management*. Addison-Wesley Publishing Company, Wokingham, England, 1988.

[Gra92] Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1992.

[Gre95] Stephen Grey. *Practical Risk Assessment for Project Management*. Wiley Series in Software Engineering Practice. John Wiley & Sons Ltd, Chicester, England, 1995.

[GS94] Robert B. Grady and Tom Van Slack. Key Lessons In Achieving Widespread Inspection Use. *IEEE Software*, 11(4):46–57, July 1994.

[Hum95] Watts S. Humphrey. *A Discipline for Software Engineering*. SEI Series in Software Engineering. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1995.

[IEE89] IEEE Std. 1028–1988, IEEE Standard for Software Reviews and Audits, June 1989. Corrected Edition.

[IEE90] IEEE Std. 610.12–1990, IEEE Standard Glossary of Software Engineering Terminology, September 1990.

[Jon96] Capers Jones. Software defect-removal efficiency. *IEEE Computer*, 29(4):94–95, April 1996.

[Jon97] Capers Jones. *Applied Software Measurement. Assuring Productivity and Quality*. Computing Series. McGraw-Hill, Inc., New York, second edition, 1997.

[Kan95] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1995.

[KKF86] B.A. Kitchenham, A.P. Kitchenham, and J.P. Fellows. The effects of inspections on software quality and productivity. *ICL Technical Journal*, 5(1):112–122, May 1986.

[Kus93] Shinji Kusumoto. *Quantitative Evaluation of Software Reviews and Testing Processes*. Dissertation, Osaka University, September 1993.

[Leh94] M. M. Lehman. Models and Modeling in Software Engineering. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 698–702. John Wiley and Sons, Inc., New York, 1994.

[McG96] Thomas McGibbon. A Business Case for Software Process Improvement. A DACS State-of-the-Art Report, September 1996. URL: http://www.dacs.com/techs/roi.soar/ soar.html.

[Mer95] *Merriam-Webster's Collegiate Dictionary*. Merriam-Webster, Incorporated, tenth edition, 1995.

[PPVW96] D. E. Perry, A. Porter, L. G. Votta, and M. W. Wade. Evaluating Workflow and Process Automation in Wide-Area Software Development. In Carlo Montangero, editor, *Proceedings of the Fifth European Workshop on Software Process Technology*, Lecture Notes in Computer Science Nr. 1149, pages 188–193, Berlin, Heidelberg, October 1996. Springer–Verlag.

[PSV95] Adam Porter, Harvey Siy, and Lawrence Votta. A Review of Software Inspections. Technical Report CS-TR-3552, Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, College Park, MD 20742, October 1995.

[PVST95] Adam A. Porter, Lawrence G. Votta, Harvey P. Siy, and Carol A. Toman. An experiment to assess the cost-benefits of code inspections in large scale software development. In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, October 1995.

[Rus91] Glen W. Russell. Experience with Inspection in Ultralarge-Scale Developments. *IEEE Software*, 8(1):25–31, January 1991.

[Shi92] Glen C. Shirey. How Inspections Fail. In Proceedings of the Ninth International Conference on Testing Computer Software, pages 151–159, 1992.

[SMT92] G. Michael Schneider, Johnny Martin, and Wei-Tek Tsai. An Experimental Study of Fault Detection in User Requirements Documents. *ACM Transactions on Software Engineering and Methodology*, 1(2):188–204, April 1992.

[Vot93] Lawrence G. Votta Jr. Does Every Inspection Need a Meeting? In David Notkin, editor, *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 107–114. ACM Press, December 1993. Appeared as ACM SIGSOFT Software Engineering Notes 18(5), December 1993.

[WBM96] David A. Wheeler, Bill Brykczynski, and Reginald N. Meeson, Jr. *Software Inspection: An Industry Best Practice*. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[Wel93] Edward F. Weller. Lessons from Three Years of Inspection Data. *IEEE Software*, 10(5):38–45, September 1993.

[Zus94] Horst Zuse. Complexity Metrics/Analysis. In John J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pages 131–165. John Wiley and Sons, Inc., New York, 1994.

Document Information

Title:	A Measurement Frame-work for Software Inspec-tions in the Quasar Context
Date:	September 1, 2003
Report:	IESE-118.03/E
Status:	Final
Distribution:	Public